Techniques for planning computer animation

John, Nigel William

Award date:
1989

Awarding institution:
University of Bath

Techniques for Planning Computer Animation

submitted by

Nigel William John

for the degree of Ph.D of the

University of Bath

1989

N. W. John

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

"Everything should always be made as simple as possible, but not simpler",

Albert Einstein

Table of Contents

Summary

Finding optimal methods of producing computer animation is an active field of research and the thesis begins with a literature review of this subject. We pick out the development of techniques for planning motion in computer animation as an area for further investigation. Many of the earlier key frame and scripted computer animation systems tended to require considerable effort from the animator. With the development of systems using physical laws greater automation has been introduced, and more complex animation can be generated. The animator can argue however that he is losing fine control over the motion produced. We want to develop a system that gives the animator as much control as possible over motion planning, without the interface becoming too cumbersome to use.

A major part of the thesis contains a description of the Controller animation system that has been designed to satisfy the above criteria. Emphasis has been given to the use of kinematic techniques for modeling motion effects. These are fast to evaluate and help to provide an easy to use interface between the animator and the animation system. Controller produces animation for entertainment applications and we require this animation to be visually realistic. To facilitate this we examine several long-established techniques that have been employed in conventional animation. Applying these techniques in Controller enables us to 'fake' reality without having to use physical simulations.

Acknowledgements

I would like to thank all members of the graphics team at the University of Bath for their help and suggestions during the tenure of my research. In particular I wish to acknowledge the assistance given to me by my supervisor, Phil Willis, and by my colleagues, Andy Hunter and John Spackman. My thanks also go to the Science and Engineering Research Council for funding the project.

I would also like to take the opportunity of thanking my parents for their moral support, but most of all for the encouragement (and proof reading) given by my wife, Shirley.

Chapter 1

Background

1.1 Introduction

This thesis is concerned with the art and science of computer animation. To introduce this subject we will look at the background of animation in general, beginning with an explanation of just what animation is.

Definition: To *animate* an object is to give apparent life to that object; movement is usually the essence of such animation.

This is one possible definition of animation and it can apply in several contexts. Most people, however, think of animation in terms of drawn animation. Here a series of gradually varied drawings (or frames) are photographed onto film. When the film is projected at an appropriate rate the figures in the drawings will appear to move*, because the retina in the human eye continues to register an image for a brief period after that image has been removed; a phenomenon known as *persistence of vision.* Presenting several images in quick succession results in them being blurred into a single continuous image giving the illusion of motion. Not just movement but anything that can change over time, colour for example, can be animated.

Traditionally animation sequences have been drawn by hand and we will give an overview of that process in this chapter. Over the last two decades, however, computers have become more and more prevalent in the creation of animation. They assist in the production process and are a valuable tool in the

* To provide an illusion of continuous motion at least 15 frames per second are required. Film, for example is projected at 24 frames per second, and video at 25 frames per second.

making of complex animation sequences. A literature survey of computer animation outlining areas of current research in this field will also be presented. From this we can identify new research possibilities, and the remainder of the thesis will describe the work that has subsequently been carried out. First we will look at the motivation for producing animation and give a synopsis of its history.

1.2 The Application of Animation

Animation has several useful applications and some examples of these are given here.

1.2.1 Scientific Simulation

Animation can depict processes that cannot be visualised by live action simulations. This makes it ideal for providing a visual simulation of a problem and thus help to solve or explain that problem. Some examples include simulations of chemical reactions, crashes, and explosions.

1.2.2 Entertainment

One of the main uses of animation is to entertain people. Commercials, film special effects, and children's cartoons all utilise animation for this purpose. The production of animated cartoons is often referred to as *character animation.*

1.2.3 Mass Communication

Governments and industry will often use animation as a means of mass communication. Animation can present a message clearly and emphasise important points.

1.2.4 Education

Animation can simplify complex processes and so is a useful tool in education. For example, it can help to explain functions of the human body such as the blood circulatory system.

1.3 The History of Animation

Animation as an illusion of movement can be traced back to the cave drawings of Neanderthal man. He would often draw animals with blurred or multiple limbs to try to give the impression of movement. Animation as we know it today, however, developed much later beginning with the invention of image projectors. The first example of such a device was the *Magic Lantern* built by Kirscher in the seventeenth century. It projected a hand drawn slide using a candle or reflected sunlight as its light source.

The next advancement was to project images in motion but this did not occur until the nineteenth century. In 1824 Peter Mark Roget published his studies on the persistence of vision. He discovered four basic principles about this phenomenon:

(i) the viewer's vision must be restricted;

(ii) the eye blurs many images into one;

(iii) a minimum speed of presentation is required to prevent the images being broken up;

(iv) a large quantity of light is required for convincing results.

This led to the invention of devices such as the *Phenakistiscope*, *Thaumatrope*, and *Zoetrope*. The Zoetrope, for example, consisted of a revolving drum with regularly spaced slits along its sides. Drawings were held on the inner wall of the drum and would appear to move when viewed through the slits. Devices such as these became popular and were a standard feature in Victorian penny arcades where they could be used, for example, to discover *What the Butler Saw*.

The development of film in the early twentieth century really led to animation taking off as a new art form. In Europe and particularly in America

various cartoon films began to appear both for instructional and entertainment purposes. In 1914 the technique of *cel animation* was introduced by Earl Hurd. This involved drawing backgrounds and characters on sheets of celluloid that could be overlayed to form the final frame. These cels are reusable and thus reduce the amount of drawings required. In 1928 Walt Disney arrived on the scene and he turned animation production into a commercial industry. His studio produced such classics as *Snow White and the Seven Dwarfs* and *Fantasia.* These films were two dimensional cartoons but, owing to the many techniques developed by the studio animators, were very convincing. These techniques are still in use today.

The production of hand animation has always been costly in terms of both time and money. A Disney production, for example, would take at least three years and over two million drawings to make. Matters were mot helped by the development of television and the resulting decline in cinema audiences. To make animation production more economically viable computers began to be introduced. Initially they were used to assist in the animation process but today they can generate the whole animation sequence. In particular, they make three dimensional animation more readily available. Productions such as *Luxo Jnr* and *Tin Toy* by the American company *Pixar* demonstrate the current state of the art. These films are still time consuming to produce as computers are not yet powerful enough to produce high quality three dimensional animation in anything like real time. The only real time applications of computer animation to date have relied on special hardware. They include extremely costly visual simulators, and computer games where the quality of animation used is limited.

For more details on the early history of animation refer to Madsen (1970).

1.4 An Overview of Traditional Animation

In this section we will review the process of traditional animation. This process dictated the early development of computer animation and many of the techniques learnt here are relevant to the computer medium.

1.4.1 The Production Line

A large team of people performing a variety of tasks are needed to produce an animation sequence. Thousands of drawings will be flowing through the production line and these have to be well organised. Different animation houses use slightly different production methods but they all follow the same basic steps. These are summarised below, and further details can be found in Magnenat-Thalmann and Thalmann (1985), and White (1986).

(i) The Script

The subject matter is extensively researched particularly when simulations are being produced. An animator, for example, will often study the structure and timing of live action movements. This will help ensure that the final animation is accurate. A detailed scenario is then prepared as a script. It will concentrate on the visual action rather than dialogue as action is more important in animation.

(ii) The Storyboard

Using the script a series of drawings depicting the key points in the action are produced as a storyboard. This visual presentation of the scenario will often identify areas where the story requires the addition of more polish. The storyboard is usually divided into *action sequences* that are in turn divided into *scenes.*

(iii) The Soundtrack

The dialogue and key music are recorded at this point as the animation must be drawn to synchronise with its soundtrack. The soundtrack is analysed phonetically to determine the precise frame position of each sound. This information is recorded on the *bar sheet.*

(iv) Designs

Visual interpretations of the characters to be used are created. *Model sheets* depicting the characters in different poses can then be produced.

(v) Animation

The first step in producing the animation is to create a *leica reel.* This is a filmed storyboard that can be projected in sequence with the soundtrack. Pencil drawings of each scene are then filmed. The animator will draw in the *key frames* or *extremes* of the action using the bar and model sheets as a guideline. Assistant animators and in-betweeners will then draw in the frames required between these extremes. When the line test is acceptable, it is cut into the leica reel. Note that all the drawings must be cleaned up at this point to ensure that a consistent style is maintained.

(vi) Trace and Paint

Each drawing will be traced or xeroxed onto a celluloid or acetate *cel.* These cels will then be coloured as appropriate and carefully checked for errors. Backgrounds are also prepared in this manner.

(vii) Final Shoot

The artwork must then be filmed. A machine called a *multiplane* (or *rostrum camera*) is used for this purpose. It consists of a series of glass layers mounted at different distances below a camera. Each cel needed for a particular frame will be placed on the appropriate glass level and the composed frame is then photographed. The multiplane allows the cameraman to achieve effects such as zooms, pans, tilts and spins. He is given a *dope* or *exposure sheet* containing the shooting information for each scene so that he knows when such an effect is required. Another machine called an *optical printer* can be used to provide fades and multiple images.

(viii) Post Production

After processing, the raw film, or *rushes*, is projected and viewed. If satisfactory the rushes can be cut into the final film, otherwise reshooting will be necessary. Meanwhile the voice track, music and sound effects are mixed onto the final soundtrack. This is then merged with the final cut of the animation film to produce the *answer print.* The animation will now be complete.

1.4.2 The Art of the Traditional Animator

Much material is available to the animator to teach him the art of animation (Madsen 1970; Thomas and Johnstone 1981; White 1986). His skill and experience in applying this knowledge will determine the overall effectiveness of the animation produced. Note that with character animation the main concern is to entertain the audience and so the action only needs to look right in an artistic rather than in a scientific sense. Several principles have been developed to assist the animator in doing this. These *principles of animation* are his tools of the trade and are summarised below.

(i) Staging

Make sure that the action is well laid out and prevent the audience from getting confused.

(ii) Straight Ahead Action and Pose to Pose

Animating from *pose to pose* is another name for key frame animation. The animator will draw poses of a character at some key frames and his assistants will then draw in the inbetweens. The resulting animation has clarity and strength. Alternatively, more spontaneity can be obtained by animating *straight ahead.* This involves drawing successive frames 'on the fly' using only the storyboard as a guideline. A combination of these two techniques is usually used.

(iii) Slow In and Slow Out

Space successive frames to make the moving object slow down or speed up to ensure smooth motion transitions.

(iv) Anticipation

Let the audience know what is about to happen by using a preparatory move. For example, swing a leg backwards before kicking a ball.

(v) Timing

The timing depends on the number of drawings being used for an action. For example, timing can be used to emphasise the weight and size of a moving object. The whole subject of timing for animation is discussed in detail by Whitaker and Halas (1981).

(vi) Arcs

Motion will look less mechanical if the path of the moving object traces out a curve rather than a straight line.

(vii) Follow Through and Overlapping Action

Make sure that an action does not end suddenly and determine if it will affect any later action.

(viii) Secondary Action

Enhance the main action with smaller secondary actions.

(ix) Squash and Stretch

Deform a moving object in order to remove the appearance of rigidity. This applies to both rigid and soft objects. Clothing, for example, should be deformed as the person wearing it moves.

(x) Exaggeration

Exaggerating an action can help it appear more realistic, or at least caricature reality.

(xi) Solid Drawing

The animator must be able to produce solid drawings of the action at a particular frame from any angle. If drawn well, extra weight, depth and balance will be added to the animation.

(xii) Appeal

It is important to give a drawing appeal to capture and maintain the attention of the audience. A quality of charm, a pleasing design, simplicity, communication and magnetism are some of the factors that can give appeal to a drawing.

1.5 An Overview of Computer Animation

The term *computer animation* is applied to any of several uses that a computer can have when producing animation. In this overview of computer animation we will distinguish between *computer assisted* animation and *modelled* animation (Magnenat-Thalmann and Thalmann 1985). This is a simple way of classifying the subject but will introduce most of the relevant topics. A more detailed survey of the classification of computer animation is given by Pueyo and Tost (1988).

1.5.1 Computer Assisted (or Key Frame) Animation

Computers were first introduced into animation production to help reduce the cost and tedium involved in the manufacture of two dimensional character animation. One possible use of a computer, for example, is to control a physical device such as the rostrum camera (Kallis 1971). More often, however, computers are utilised in the key frame animation process where they have two main functions:

(i) the drawing and colouring of key frames;

(ii) automatic inbetweening.

Note that the actual image in each key frame has to be interpolated, a process called *shape interpolation* (Zeltzer 1985) or *image based key frame animation* (Steketee and Badler 1985). Lewell (1985) provides an overview of how the Hanna Barbera studios use computers in their animation process.

Key frame animation can also be extended into three dimensions by using a technique called *parametric key frame animation* (Hanrahan and Sturman 1985; Steketee and Badler 1985; Zeltzer 1985). Here an object is described by a set of parameter values such as its position vector. Each frame will be constructed using the information supplied by these parameters. The animator will use either an interactive system or an animation programming language to set the value of each parameter at the key frames. The computer will then interpolate the parameters (rather than the shape of the object) to produce the inbetween frames. Interpolating between the physical parameters of a body will produce better results than those obtained from shape transformations.

1.5.2 Modelled (or Algorithmic) Animation

The computer should not just be a labour saving device, it should also improve the quality and complexity of animation. Drawing and manipulating objects in three dimensional space, for example, would be a difficult procedure without the aid of a computer. This is also true of providing animated simulations from numerical data. Applications such as these where the computer is the main tool of animation production are often termed modelled animation. The motion applied to the objects here is usually described algorithmically using physical laws, for example. The animator is thus relieved from tedium of producing thousands of drawings and allowed to be more creative.

The production of modelled animation can be divided into four stages (compare these with the stages involved in traditional animation production):

(i) modeling objects in three dimensions;

(ii) animating the objects;

(iii) rendering the frames;

(iv) post production.

The emphasis placed on each stage will vary, depending on the application of the animation. Each stage also has its own set of problems and we will present details of these in the next section.

1.6 Computer Animation: The State of the Art

Computer animation must at least match the results obtainable from traditional methods and should allow more complex animation to be created. Merely attempting to emulate traditional animation is not enough (Thomas 1984); a new generation of animation should be possible. The computational power of computers provides us with the basis to achieve this aim and realistic static images can already be produced. We need to extend this use of computers further to include the temporal changes required for animation.

This section presents a literature survey of the current state of the art in computer animation. For clarity, we have divided the subject into separate topics although these should not be regarded as disjoint. As we are concerned with a wide ranging field there will often be an overlap between them.

1.6.1 Generating Key Frames

Burtnyk and Wein (1971) and Catmull (1979) describe typical image based key frame animation systems. With such a system the animator will draw a set of key frames directly onto a computer graphics screen using a data tablet. Software tools are often provided to aid in the drawing of these key frames enabling the animator, for example, to fit smooth curves. Paint systems are another common feature. The area filling algorithms provided by these alleviates the tedium of hand colouring the frames. A key frame can also be built up by composing several images that the animator has already drawn (Wallace 1981). Unlike cel

animation many images can be composed without any loss in the quality of the key frame.

The animator will have to identify a set of boundary points (or vertices) on each key frame. The computer can then generate inbetweens by interpolating between corresponding points in successive key frames. Burtnyk and Wein (1976), for example, record the order of the strokes that the animator makes while he is drawing a key frame. A correspondence is then set up by making a stroke to stroke mapping between successive key frames.

Automatic inbetweening using shape interpolation, however, has its limitations (Catmull 1979b). A key frame image is a two dimensional projection of a three dimensional character as visualised by the animator. Often some part of the character will be obscured by another part of it, for example, if a head is drawn in right profile then the left ear will not be visible. The computer does not have any information about this obscured part and so it has to be supplied in some way, usually by the input of more key frames. This means that the time saved by using a computer in the first place is being negated. Catmull also points out that efficient handling of the thousands of frames that are created during the animation process is not often considered. He advocates greater use of data base technology for this purpose.

A possible solution to the missing information problem is to define the key frames in three dimensions. Geometric models of the characters have to be provided so that the computer will have all the information it requires. Two dimensional projections of the frames can then be made after the interpolation process. First, however, methods of three dimensional modeling are needed.

1.6.2 Modeling in Three Dimensions

Both parametric key frame animation and modelled animation take place in simulated three dimensional environments. The computer models required here must also be defined in three dimensions and Lansdown (1983) reviews many of the modeling techniques used for this purpose. They include rotational sweeping,

curved surface representations, quadrics, and stochastic methods. Note that the format of the model used will often determine how the motion of an object can be calculated. A model format that will facilitate animation is therefore advantageous.

The use of wire frame models provide the simplest three dimensional representation. Such models have low storage requirements and are fast to calculate. Solid models, however, provide the realism needed for most animation applications. Ostby (1987) identifies two classes of geometric solid modeling:

homogeneous: all elements of the model are expressed as collections of some basic primitive such as polygons. These are simple to use but are a crude way of expressing non-polygonal objects;

heterogeneous: all elements of the model are expressed as collections of wider-ranging primitives such as quadrics, or patches. Quadrics are easily parameterised (a property that facilitates animation) but limited in what they can express. Conversely, patches give good approximations of objects but can be difficult to parameterise.

For flexibility a combination of techniques should be offered.

The use of polygonal models in key frame animation is similar to the two dimensional case (Thalmann 1989) and a correspondence between vertices at each key frame must be defined. Extra vertices often have to be added to ensure that the same number of vertices exist at each key frame. If faceted models are being used then the process is more complex as there must be a correspondence between facets as well as vertices. More often three dimensional interpolation is based on parameterised models using joint angles, for example. Whatever type of model is used, however, an object creation facility is required.

The animator will typically be provided with an interactive graphics editor such as the "body-building" system (Magnenat-Thalmann et al. 1985). An editor of this type will allow the instantiation of elemental primitives that have been

transformed by some combination of translations, rotations, scales and shears (Herbison-Evans 1978). Instances of already modelled objects can also be utilised by assigning them different scales and colours (Magnenat-Thalmann and Thalmann 1985b). The primitives are often composed using boolean operations (unions, intersections, etc) as, for example, in constructive solid geometry (C.S.G). Another technique is to build the models hierarchically. This is particularly useful for articulated objects as it has the advantage that any movement applied at one level of the hierarchy will automatically effect everything below this level. Also, when an object is distant from the viewer there is no need to display it in as much detail as when it is close up (Lansdown 1983; Pueyo and Tost 1988). A hierarchical model can be used for displaying an object at the level of detail required.

It may be possible to write a computer program to build object models and this is more appropriate when an object is to be constructed out of many constituent parts. Some computer proficiency is necessary here, however. A different approach is to reconstruct a three dimensional model from digitised photographs of the real thing (Magnenat-Thalmann and Thalmann 1985). This technique is useful when an object is too complex to model using other methods.

Stochastic methods such as fractals are used in solid modeling to create objects that exhibit a degree of randomness. Trees and mountains are well known examples of the application of fractals. Stochastic methods can also be applied in animation for modeling fire, water and clouds where movement can be regarded as a parameter of the model. Reeves (1983) has described a method of modeling such 'fuzzy' objects using *particle systems*. A particle is represented as a short, anti-aliased line segment whose lifetime and movement are controlled by stochastic procedures. Clusters of these particles defined in the appropriate manner are used to create the desired object. This work was later extended (Reeves and Blau 1985) by using structured particle systems to model solid objects of trees and grass. This technique enables the production of complex motions with random variation such as a field of grass blowing in a breeze.

Note that the quality of the models generated for character animation will usually be to a higher standard than that used in scientific and educational applications. Appearance and visual richness are of more importance in the former case.

1.6.3 Inbetweening in Key Frame Animation

Once a correspondence between key frames has been defined, the next step is for the computer to calculate the inbetweens. Whether for two or three dimensional applications the computer has to define a path between corresponding points in the key frames, taking both space and time into account. Several interpolation methods are available to do this, most of which can be applied to either vertices or parameters. Note that motion, particularly that of living beings, is complicated to model accurately and often appears artificial or robotic. The result obtained will be greatly determined by the interpolation method used. When evaluating such methods Reeves (1981) considers their generality, smoothness, efficiency, and ease of specification. These criteria are also considered for the methods described below.

In linear interpolation the corresponding points between each pair of key frames are joined by regularly divided straight lines. This is the simplest and fastest method of inbetweening but produces the least favourable results. At the key frames there is often a lack of smoothness in the motion, discontinuities in speed, and distortions in any rotations used (Thalmann 1989). Using fewer key frames will of course produce less discontinuities but will make it more difficult for the animator to define animation in any detail. Non-linear divisions of time will produce acceleration effects and are simple to introduce (see chapter 3). The motion will still be in a straight line, however, and can look artificial. Techniques of providing a non-linear spatial division also need to be applied.

Burtnyk and Wein (1976) introduce a skeleton technique to ease inbetweening and allow the animator to define complex motion. An initial correspondence is made between a skeleton and a fully drawn figure. The animator then only needs to animate the skeleton with the details of the figure

being added later. Such a skeleton will always have a similar structure in the key frames and so better inbetweens can be calculated. The approach adopted by Gomez (1984) also attempts to simplify inbetweening. Instead of having arbitrary spaced key frames he defines them only when something happens to an object. These *events* are stored in a linked list called a *track*, a separate track being used for each object (or moving part of an object). A frame is then formed from the union of activity of all the tracks.

Baeker's picture driven animation system (1969) introduced parametric curves, called *P-curves*, for defining motion. The shape of a P-curve represents the trajectory (in two dimensions) of some moving point. A trail of symbols rather than a continuous line is used to plot this shape. These symbols are equally spaced in time and so the relative density of them along the trajectory will indicate the speed of motion. Both spatial and temporal information are thus available on a single graph.

Reeves (1981) use of *moving point* constraints in the inbetweening process takes a similar approach to the P-curve method. A moving point is a curve sketched by the animator to connect a pair of corresponding points in two successive key frames. The shape of this curve determines the trajectory of the motion, and symbols marked at regular intervals of time along it determines the timing of the motion. As the trajectory is not linear a better approximation of natural motion will be obtained. The shape of the object at each frame is then obtained by interpolating through the moving points using an appropriate curve drawing algorithm. This method enables multiple paths and speeds to be specified and helps to reduce the discontinuities at the key frames. The animator, however, is required to specify additional information other than just the key frames.

Piecewise cubic polygons (or splines) can also be used to connect corresponding points in key frames. Kochanek and Bartels (1984) describe such a method. They allow the shape of the spline to be adjusted at the key frames by altering tension, continuity and bias parameters. The animator can thus fine tune the movement of an object without having to redraw key frames. This method

can be used to interpolate angles and vectors and Thalmann (1989) makes use of this to animate the human body.

Steketee and Badler (1985) give an interpolation method for parametric key frame animation. They make use of the B-spline curve representation as it provides second order continuity and therefore smooth curves (see chapter 3 for more details). Their system allows for kinematic control and the joining and phrasing of successive motions. They claim, however, that there is no satisfactory solution to the deviations between the interpolated image and the object being modelled.

A different approach for three dimensional applications is to draw smooth path curves through space to control the interpolation process. Shelley and Greenberg (1982) also utilise the B-spline curve representation for this purpose. The animator first defines a B-spline in three dimensions to represent the trajectory of the object or camera. A separate B-spline is then used to provide the timing information. This second curve is regarded as a function of velocity against distance and the animator has to input the required velocity values along it. The use of path curves such as these enable the animator to think about the entire motion of an object. Unlike key framing, however, it is difficult to visualise the total configuration of the animation at a given time. Matters are also more complicated if the object has to exhibit internal motion. The post process techniques described by Lundin (1984), however, provide one method that can be used to solve this problem.

Spencer-Smith and Wyvill (1989) also control motion by using a spline. They have developed a *four dimensional spline* that passes through both space and time. The way in which the motion timing is defined is again similar to the P-curve approach. The spline is plotted using a trail of spheres that are equally spaced in time. The relative density of these spheres and hence the motion effect achieved can be varied by adjusting the control points of the spline.

The introduction of cubic splines has helped to provide motion with a smoother, more natural appearance. The animator has to define or control several

additional parameters, however, and so using splines can be a time consuming process. If realistic motion is the major concern then a different approach is to utilise physical laws. Brotman and Netravali (1988), for example, describe an interpolation process that makes use of differential equations obtained from classical mechanics. The application of physical laws is more common in modelled animation processes, however.

1.6.4 Motion in Modelled Animation

In three dimensional animation actors, cameras and lights are generally defined at each frame by an appropriate set of parameters. For example,

camera = {*location*, *direction*, *zoom*};

actor = {*location*, *orientation*};

light = {*location*, *direction*, *intensity*}.

Animation is obtained by gradually varying the value of these parameters across successive frames, usually by applying a list of transformations. To achieve motion around a set, for example, the parameter that defines the location of each object must be updated using an appropriate translation. For cameras, effects such as spins, pans and tilts must be allowed for (Magnenat-Thalmann and Thalmann 1986). The simulation of different lighting effects should also be possible. Methods of calculating the change in state of these various parameters are therefore needed. In modelled animation the two techniques most often used for this purpose are:

(i) kinematics;

(ii) physical laws such as the laws of dynamics.

Kinematic motion is obtained by calculating positions, speeds and accelerations as a function of time. We have already seen how kinematics have been employed in the key frame animation process (§1.6.3). Linear interpolation, spline interpolation and path curves all fall into this category. Often the motion

applied to an object is *faired* (Lansdown 1983). Here the object is made to accelerate from rest at the beginning of its movement, and decelerate to rest at the end of its movement. This emulates the traditional animation principle of slow in and slow out thus providing a smoother motion effect. By using techniques such as fairing the animator can obtain convincing motion in his animation. Sometimes, however, the motion is too laborious or too complicated to model kinematically and so physical laws are utilised.

The use of dynamics to model motion, particularly that of the human body, has been developed (Armstrong and Green 1985; Wilhelms and Barsky 1985). With dynamics the forces and torques acting on an object are taken into account. This dynamic information is then used to calculate the kinematic motion of the object. Newton's laws of motion are the basis of dynamic modelling (Wilhelms 1987; Selbie 1989). For example, Newton's second law can be stated as

$$\mathbf{F} = m\mathbf{a} \qquad (*)$$

where **F** is the force acting on an object, m is the mass of the object, and **a** is the acceleration that the object will undergo. Numerical methods are usually needed to satisfy the constraints of the animation but these are derived from (*). The actual forces used by the system can be calculated automatically (as with gravity), modelled with springs and dampers, or supplied by the animator.

Hahn (1988) and Miller (1988) show how dynamics can be used to produce realistic animation. Hahn's system models the motion of rigid bodies taking mass, elasticity, friction and moment of inertia into account. In his animation of snakes and worms Miller makes use of a mass-spring system where muscle contractions are modelled by spring tensions. The use of dynamics is essential in such simulations where the modeling of reality is important but there are disadvantages. Dynamic systems can be hard to implement, the computation time is higher than with kinematics, and the control of forces and torques is non-intuitive and so difficult to use.

Many authors consider that the optimal approach for modeling motion is to provide a combination of kinematic and dynamic techniques. Wilhelms (1986),

for example, describes an interactive motion control editor that achieves this. In her VIRYA system the animator will control most of the animation kinematically but have the option of using dynamics to add more realism where required. Forest et al. (1986) show how the same approach can be used in the animation of articulated bodies. A mixture of kinematics and dynamics is also used by Witkin and Kass (1988) in their *spacetime constraints* method of animation. The animator specifies a minimal number of kinematic constraints such as an object's location at different times, and the manner in which it is to move. Physical laws are then used to calculate how best to satisfy these constraints. By satisfying kinematic constraints in a physically valid way they claim that traditional principles such as anticipation and timing will emerge automatically. The approach of Pintado and Fiume (1988) is different again. Their motion specification and control environment system based on fields is a kinematic technique. The *dynamic splines* that they develop, however, mimic the effects of dynamic control but without the high computational cost usually associated with this.

Other methods of modeling motion can be used. Magnenat-Thalmann and Thalmann (1985c) advocate greater use of *evolution laws* that change the state of some system over time. Brownian motion and chaotic attractors are two examples of such laws. In a similar vein Wilhelms (1987b) argues that the control of motion can be aided by integrating knowledge from other fields such as robotics, biology and physics. Use of more automatic techniques is also advocated to counteract the trend towards greater complexity. Algorithmic control, for example, is useful for modeling repetitive motion. The motion is generated using a series of preprogrammed instructions and so little user input is required. Applications of this technique include modeling the elliptical orbit of a planet and the oscillation of a pendulum.

Using a technique called *inverse kinematics* it is possible to determine the orientation of an object given the position of some distal part of that object. The programming of a robot arm for object grasping is an inverse kinematics problem and is analogous to object grasping in human animation (Arnaldi et al. 1989).

Another of its uses in animation is to ensure that a body maintains realistic contact with the ground. The inverse kinematics problem can be divided into two stages:

(i) find any solution to achieve the desired goal;

(ii) find the best solution.

As the complexity of an object increases so does the difficulty of solving the problem. An analogous problem is that of *inverse dynamics* where the forces and torques that satisfy certain constraints must be found. Barzel and Barr (1988) use inverse dynamics in their modeling system. The constraint forces calculated cause the assembly of the model components and ensure that the components stay together as the model moves.

Another recent development is behavioural control where environment interactions are taken into account. The motion of an object here is made to depend on the behaviour of other objects. The system has to recognise the state of the environment and generate a response to it. Reynolds (1987) describes a model of polarised, non-colliding aggregate motion that can be used to generate flocks and herds. His method is an elaboration of particle systems using birds say, instead of particles. The behaviour of each bird is simulated independently. They try to both stick together and to avoid collision with each other or other environmental objects. Collision avoidance is another problem from robotics that is applicable to computer animation. When defining path curves, for example, the animator could easily end with objects that interpenetrate owing to the proximity of two or more paths. The animation system should be able to detect such collisions and respond to them in an appropriate way. Moore and Wilhelms (1988) suggest methods of solving this problem.

1.6.5 Human Animation

The realistic animation of computer generated human figures is a complex problem and is a large area of research in computer animation. We present a summary of the work carried out in this area whilst more detailed surveys can be found in Badler and Smoliar (1979), Magnenat-Thalmann and Thalmann (1985) and Tost and Pueyo (1988).

The first step is to produce a three dimensional representation of the human body. The methods generally used for this are:

stick figures: a skeleton consisting of a hierarchical set of limbs connected by joints (Zeltzer 1982). A stick figure is easy to store and manipulate but it is difficult to represent some motions such as twists;

surface models: a surface 'skin' representation is obtained by surrounding a skeleton with planar or curved patches. A more realistic model results but it is tedious defining the patches and computationally expensive to render;

volume models: the body is decomposed into primitive volumes such as ellipsoids (Herbison-Evans 1978) or spheres (Badler et al. 1970). Not as realistic in appearance as surface models but far better than stick figures. Also, as they are easy to define geometrically, efficient hidden surface algorithms can be used to speed up visualisation (Herbison-Evans 1982).

Most systems will use stick figures for defining motion with the surface or volume model being applied later. Magnenat-Thalmann and Thalmann (1987) detail the steps involved when creating a synthetic human based on a real life person.

Owing to the complexity of the human body the specification of its movement is not an easy task. A possible solution is to utilise choreography notations such as *Labanotation, Eshkol-Wachmann notation,* and *Benesh notation.*

These have been developed to record dance scores accurately, a problem of representing three dimensional movements on paper. Labanotation views the body as a set of joints connected by limbs while in Eshkol-Wachmann notation limbs are connected at joints. Both of these notations can be used with the stick figure representation, for example, Calvert et al. (1980) have used Labanotation. Benesh movement notation is a hierarchical method that uses a music-like stave for recording choreography. Singh et al. (1983) have developed an interactive editor for such notation but this method is not used in computer animation as often as the others are. The *Effort/Shape* notation (Badler and Smoliar 1979) should also be mentioned. This notation is based on a muscular representation of the body and so dynamic characteristics of movement can be specified.

Having obtained a motion description it must be applied to the human figure that is to be animated. We have already looked at key framing methods and indicated some of their applications to human body animation. Forest et al. (1986b) control motion by specifying joint angles for the human figure at each key frame. Kinematics or physical laws are used to interpolate the joint angles according to the requirements of the animator. With systems of this type the flow of motion is determined by the interpolation technique used. The ability to add refinements to the resulting animation is therefore essential. Kinematic and dynamic transformations can also be used to move a figure along a path and this technique is useful for making global motion specifications. The main problem here, however, is in describing internal movement of the body such as walks. Armstrong et al. (1986) apply forces and torques to the limbs of the human figure and so internal motion is automatically specified. They supply an interactive interface to allow the animator to control and adjust the forces applied.

To take some of the burden of motion specification for articulated bodies away from the animator, *goal-directed* motion techniques have been developed (Korein and Badler 1982; Zeltzer 1982b). Using this approach the animation software generates most of the motion and the animator just has to specify the end constraints he requires. The cost is a trade off with the animator's artistic control. Another technique that can be used is *rotoscopy*. Here human

movements are digitised from real life and applied to their synthetic counterparts.

The ability to vary the anthropometry of different human figures should also be available. This facility is useful in simulation applications where, for example, the effects of a car crash on humans with a variety of different builds might need to be determined. Grosso et al. (1989) describe a system offering such a facility. They use a spreadsheet-like interface to alter the body parameters of a human figure. Adjustments to the anthropometric structure of a figure can also be used to add more 'character' to it.

The animations of faces and hands are particularly complex and usually carried out separately from the rest of the body. These entities are generally modelled as surface patches to which different motions can be applied. One approach for animating a face is to associate two types of parameters with the surface patches (Parke 1982). The *conformation* parameters define a neutral face that is altered using *expression* parameters. Alternatively, a facial model based on muscle deformations has been developed (Waters 1987). Emotions and speech are two important aspects that should also be taken into account. A survey of the more important facial animation techniques can be found in Magnenat-Thalmann (1989). With hand animation, skeleton motion and shape deformation must be considered. We have already mentioned that hand grasping is an inverse kinematics problem.

1.6.6 Computer Animation Systems

One way of considering the data flowing through an animation system is in terms of the *degrees of freedom* of the moving objects (Zeltzer 1985; Wilhelms 1987b). For example,

particles: particles have three degrees of freedom (a translation in three dimensional space);

rigid bodies: a rigid body is made up of several points that must move as one. They have six degrees of freedom (a three dimensional translation and a three dimensional rotation);

flexible bodies: a flexible body contains an infinite number of points that can move relative to each other. The body is approximated with a finite number of control points, each having three degrees of freedom that vary over time;

articulated bodies: the total degrees of freedom is equal to the sum of the degrees of freedom at each joint.

Having fifty or more degrees of freedom involved at each frame is not unusual. Considering the many frames required to produce even a short animation sequence a large volume of data has to be handled by the animation system. Note that there may also be constraints applied to the degrees of freedom. A human head, for example, can be rotated but not through 360 degrees. The animator requires a system that will enable him to control these data efficiently.

Several different types of animation systems have been developed. Zeltzer (1985) states that an animation system will fall into one of three categories depending on how the motion specification is dealt with. These are:

(i) guiding systems;

(ii) animator-level systems;

(iii) task-level systems.

An integration of these systems may also be advantageous.

In *guiding systems* animation is usually defined and created at a graphics screen. These are interactive systems and so provide an immediate response on what the animation will look like. The animator has nearly complete control over the motion and can specify fine details. Guiding systems tend to be unsuitable for specifying complex motions, however. Computer assisted and path specification systems fall into this category and specific examples include Bbop (Stern 1983),

TWIXT (Gomez 1984) and Graphicsland (Wyvill et al. 1985).

The *animator-level* or *scripted* systems are animation languages. The computational power of a programming language is thus made available to the animator. He has to prepare a script by describing actions as low level elemental motions such as translations and velocities. Control structures enable him to associate successive iterations of a loop with movement at successive frames. Parallelism, synchronisation and data abstraction are also achievable. Examples of systems falling into this category include ASAS (Reynolds 1982), MIRA 3-D (Magnenat-Thalmann and Thalmann 1985), and NEM (Marino et al. 1985). Scripted systems often allow for *adaptive motion* whereby information from the environment is taken into account when calculating object positions. Collision detection can be built into the system in this way. The disadvantage of scripted systems is that the animator has to have a reasonable proficiency in software development.

Task-level systems rely less on the animator and more on the intelligence of the system. The animator describes motion implicitly using high level terminology such as 'walk' or 'swim'. Environment information is kept in a database or knowledge base and is used by the system when it calculates the required motion. One problem, however, is that there may be more than one solution to the animator's constraints. Goal-directed systems (§1.6.5) can be placed into this category but the development of a complete task-level system is still a matter of research. Task-level systems offer a more user-friendly interface than other types of systems and so are easier to use. The trade off, however, is with the ability of the animator to use his artistic skills.

Related to task-level systems is the use of artificial intelligence and expert systems in computer animation. Magnenat-Thalmann and Thalmann (1986b) have developed EXPERTMIRA, an animation language that uses concepts from artificial intelligence. In a similar vein Badler (1989) uses natural language to augment his task-level animation process. Arya (1986) describes the benefit of using a functional approach to implement animation. A kernel of primitives can be set up and combined into higher order functions.

A method that incorporates facilities from all the different categories of animation systems will give complete yet economical motion control (Zeltzer 1985). Interactive systems, for example, are usually better for designing motion (Entis 1986), and Hanrahan and Sturman (1985) who use a scripted approach resort to key framing for motion definition. Schlag (1986) also encourages the integration of scripting and interaction into a single system. A final example is provided by Chuang and Entis (1983). They use a key frame based system that utilises a script facility for defining relationships between objects that affect their motion. They place emphasis on the use of software tools and intermediate stage graphics for design flexibility.

1.6.7 Rendering Animation

Once the animation has been defined the object, camera, and lighting information are amalgamated to create a scene model for each frame of the sequence. A variety of methods can then be used to produce a static image of each frame, for example:

Renderer	Comments
wire frame	fastest, useful for previewing animation
scan line algorithms	fast for solid rendering, can lack detail
ray tracing	simulates optical laws, realistic but slow

The speed at which the animation can be rendered is an important consideration. Ideally the animator should be able to define and view an animation sequence in real time. None of the above rendering techniques is fast enough on conventional hardware to produce the minimum of fifteen frames that would be needed every second. The actual time taken to generate each frame will range from a few seconds to many hours depending on the power of the computer, the complexity of the scene, and the renderer being used. The rendering of a complete animation sequence is therefore a time consuming process and so methods of improving this time performance have been researched.

The term *frame to frame coherence* refers to the similarity of successive frames in an animation sequence. It should be possible to exploit this property to

reduce the time spent performing hidden surface removal calculations in scanline algorithms. Many flight simulators, for example, contain special hardware that rely on frame to frame coherence. This property is also used by Hubshman and Zucker (1982), but here only the position of the view point is allowed to change whilst the scene remains static. Noma and Kunii (1985) reverse this situation by keeping the view point static whilst the scene moves in front of it. Both these methods have further restrictions applied to them and so their use is limited.

Shelley and Greenberg (1982) have used a variation of frame to frame coherence. In their animation system each successive position of the view point lies on a smooth path defined by the animator. They use the coherence of this path to reduce the amount of culling and sorting that must be performed when determining the visible surfaces.

The time needed to ray trace an animation sequence can also be reduced. Glassener (1988) uses a hybrid adaptation of space subdivision and bounding volume techniques applied to four dimensions; with time as the fourth dimension. Information from the animation is used at a preprocessing stage to produce a *spacetime* subdivision. This subdivision is then used to determine the volume of space that will contain an object at a given time. The number of rays that need to be traced is thus reduced.

Effects that will improve animation can also be introduced at the rendering stage. A fast moving object in an animation sequence, for example, will often suffer from temporal aliasing and appear to move with a jerky action. Compare this with the blurred image of such an object that the human eye will register in a real life situation. This suggests that the motion of fast moving objects in animation will appear smoother and more realistic if they too are blurred. Several methods of achieving such *motion blur* in animation have been proposed. Korein and Badler (1983), for example, have employed temporal antialiasing and one technique they describe is *supersampling*. This involves taking several images for each frame at slightly different points in time. The intensity of corresponding pixels in all these images are then filtered to produce the frame pixel intensity to be used. Cook (1986), however, claims that supersampling can only reduce

aliasing and cannot remove it completely. His strategy is to replace aliasing with noise as noise is considered to be more acceptable to the human eye. He uses a stochastic method applied to the sampling of time called *jittering*. It involves dividing the frame time into slices and randomly assigning a slice of time to each sample point. Motion blur can also be achieved in particle systems, by using a model of an optical camera (Potmesil and Chakravarty 1983), and by using post process algorithms (Max 1989).

The traditional principles of animation should not be forgotten here. At the rendering stage, for example, effects such as squash and stretch can be introduced (see §1.4.2). Methods of distorting the shape of solid objects have been described (Barr 1984), and Bethel and Uselton (1989) have provided tools for shape distortion in computer assisted key frame animation.

1.6.8 Post Production

The animator will need to view the animation before it is photographed onto film. Although it is sometimes possible to generate and project a short two dimensional sequence in real time this is not generally the case. Frames that have already been rendered can be viewed using *real time playback*, however (Magnenat-Thalmann and Thalmann 1985). The rendered frames are placed into mass storage from whence they can be displayed on a graphics screen at the appropriate rate. Denber and Turner (1986) describe a differential compiler that will allow such real time playback on a general purpose computer. Each frame is run through a data compression algorithm and only the differences between successive frames are placed into memory. Their system also allows for frame editing.

When satisfactory the rendered frames have to be transferred onto film or video and a soundtrack applied. Unlike in conventional animation the soundtrack tends to be considered only at the post process stage. Computer animation to date has generally relied on music and sound effects rather than dialogue. The problem of synchronising the animation to the dialogue is thus avoided. This is an area requiring more research. Note that the development of facial animation is important here.

1.7 Research Possibilities

The literature survey presented in the previous section details the large volume of research that has been carried out in the computer animation field. Several areas are still in need of improvement, however.

The rendering of an animation sequence is the most time consuming stage of computer animation production. In most animation systems each frame is generated sequentially using a standard rendering technique. It should be possible, however, to utilise the frame-to-frame coherence of successive frames to greatly speed up the rendering process. We have seen that some success has already been achieved in this area (§1.6.7) but these solutions impose several restrictions on the animation. An all-embracing solution has yet to be discovered.

Substances such as skin and hair are difficult to model realistically on a computer and are areas of current research (Tost and Pueyo 1988). A skin model, for example, would probably need to be based on some form of deformable continuous surface. The modeling of facial expressions is another related area of active research. New modeling techniques designed specifically for the purposes of animation should also be evolved. At present most people generate objects using techniques taken from other areas of computer graphics. They then find that limitations are placed on the ways in which they can make their objects move. None of the techniques we examined earlier (§1.6.2), for example, overcome rigidity well and it is hard to show that living things bend as they move.

The development of motion specification systems is probably the most active area of research in computer animation. There has been a recent trend here towards the production of more realistic motion by making greater use of physical laws. Often, however, satisfactory results can be obtained without having to resort to complicated strategies. Interpolation techniques, for example, are cheap and easy to implement and should be more widely developed. Systems that offer the animator a combination of methods for motion specification are becoming more prevalent.

At the moment there is a progression towards task-level systems and away from computer assisted animation. The task-level approach injects more automation into an animation system and so makes it easier to use. The drawback, however, is that the skills of the animator are in danger of being overlooked. We should not forget that traditional animation is a well established art form that produces high quality animation. The recent feature film *Who Framed Roger Rabbit?* demonstrates this fact admirably. The art of the traditional animator should be utilised when generating computer animation. Greater use of the principles of traditional animation in the computer medium has already been advocated (Lasseter 1985; Van Baerle 1985). Lasseter emphasises that the success of character animation lies in the 'personality' of the character. By this he means that the animation principles should be applied intelligently to produce convincing, believable results. The use of the traditional principles of animation is a theme open for development in computer animation.

1.8 Summary and Conclusion

This chapter has given an overview of the production of both traditional and computer animation. We have noted that the technique used to accomplish a particular stage in the animation process will often depend on the application of the animation. Visual realism is important in entertainment applications whilst priority is given to precision and real time performance in scientific simulations. Our interests lie with the former class of application and this thesis will be concerned with techniques that are appropriate to this area of animation.

Although we have pointed out the need for improved modeling and rendering techniques in computer animation these will not be developed in this thesis. We will concentrate on finding new ways in which computer animation can be specified, particularly for entertainment applications. To ensure that the skill of the animator remains an important factor we will avoid the introduction of too much automation into the system. Instead we will encourage the animator to utilise the principles of traditional animation to achieve the appearance of reality. We do not want to make the animator's task difficult, however, and so a well

designed user interface will be important.

To summarise, we can identify two main aims for this thesis:

(i) to provide a system of animation planning that is straightforward to use but at the same time keeps the animator in control;

(ii) to enhance the animation effects attainable by incorporating the principles of traditional animation.

Chapter 2

The Controller Animation System

2.1 Introduction

The first aim of this thesis is to provide a flexible system for planning computer animation that is straightforward to use and keeps the animator in control. We will begin this chapter with an overview of the graphics environment available for the development of such a system. This includes the facilities that can be used to generate three dimensional models of the objects to be animated.

The *Controller* animation system is then introduced (see also John and Willis 1989). Its operation is based on that of a television studio with the intention of creating a user friendly interactive graphics system in which the animator becomes the programme controller. This studio model is particularly apt as we are concerned with producing animation for entertainment applications. The programming principles used to implement Controller are also important and are discussed later in the chapter.

2.2 The Graphics Environment

A determining factor in the development of any computer graphics package are the facilities that are available. For our purposes the equipment should make possible the production of three dimensional computer animation. This section details the resources available in the Graphics Group at Bath University.

2.2.1 The Processors

Two High Level Hardware Orion-1/05 super minicomputers are available exclusively for graphics work. These machines are high performance computers aimed at the scientific community. They run the UNIX 4.2 BSD operating system and thus benefit from the extensive range of programming tools and languages available for UNIX implementations. The principle programming language is C.

The Graphics Group initially used Orion-1 computers and these were comparable in power to a VAX 11/750. However, with the inclusion of the CLIPPER 32-bit microprocessor (with its built in floating point unit), the performance of an upgraded Orion-1/05 exceeds that of a VAX 8600. This computational power is useful for many graphics applications. In particular it makes feasible the rendering of an animation sequence using the realistic, but computationally expensive, technique of ray tracing.

2.2.2 Input Devices

Alphanumerical information is entered into the computer using a keyboard. However, for specific graphical input a digitising tablet and four-button cursor (or puck) is used. Magnetic induction generated between a coil in the puck and grid wires inside the tablet allows the position of the puck to be detected. The puck coordinates are then transmitted to the computer together with an indication of whether a button is being pressed. The puck is a locator device for obtaining screen coordinates and the buttons are choice indicators.

A colour digitiser is also available for scanning images into the computer. These images can be used as texture maps in the modeling stage.

2.2.3 Output Devices

Each computer has an eight-bit colour display with a resolution of 1280 by 1024 pixels for graphical output. The images displayed on this screen can also be captured using the Graphics Group's Colour Graphic Recorder. This is a device for recording colour photographic hard copy from the analogue video output of a raster scan screen. We can use it to obtain 35mm still photographs or, more appropriately for animation purposes, 16mm cine.

A standard visual display unit used with the keyboard is, of course, also available.

2.2.4 The Software Environment

Software in the group is written in the C programming language. It includes a library of graphics operations and a variety of tools used with the above hardware. As well as providing the interface between the screen and the tablet, this library offers drawing facilities, colour control, and menu handling facilities.

A highly interactive system with good quality raster scan colour display is thus provided, encouraging interactive rather than batch-oriented programs to be written.

2.2.5 Ray Tracing and Wire Frames

During the development of our animation system a concurrent research project has taken place to improve the performance of ray tracing algorithms (Spackman 1989). As a result of this project, an efficient ray tracer is available that can synthesise realistic solid images containing many optical effects. Surface radiance, shadows, reflection, refraction and light attenuation are just some of the features that are offered. Obtaining such realistic effects in a computer animation is desirable, and so we decided to utilise this system. Note, however, that the computational cost of generating a ray traced image is high, and it can several hours to render a complex image at a reasonable resolution.

A scene is described in a left-handed coordinate system and is divided into a a lighting model, and an object model (see appendix A). The lighting model defines the configuration of the viewer, and the position and intensity of all the point light sources required. The object model describes the bodies that are to be found in the scene. They are assembled from cones, cubes, cylinders, ellipsoids, planes, spheres and tori, using constructive solid geometry (unions, intersections and differences). The objects so formed are then defined optically by reference to the materials from which they are made. Silver, for example, can be simulated by making the colour of the material an appropriate shade of grey and giving it reflective properties. Texture maps can also be wrapped around specified objects.

A facility to assemble three dimensional models interactively has not yet been implemented. The design process can therefore be cumbersome, especially if the model is complex. Fortunately, however, each model can be previewed before it is rendered by the ray tracer. The same two source files used by the ray tracer can be fed into a mesh renderer producing a wire frame representation of the model. For speed, the mesh renderer does not do any constructive solid geometry or hidden line removal. The resulting image, however, is usually informative enough for confirmation of the model design.

The models generated will be needed at various stages throughout the production of an animation sequence. For example, they will be used when converting the data specified by an animator into scene models for each frame of the animation sequence (see chapter 4).

2.3 Designing Controller's User Interface

The art of animation is in making objects move convincingly and traditionally this has depended on the skill of the animator. With the development of computer animation systems the resulting animation is also dependent on how the system allows motion to be planned. The design of the user interface is therefore an important factor in determining the quality of the animation produced and should be well thought out before proceeding to the implementation stage. We will begin this task by determining exactly what is required from our system of

animation planning (hereafter referred to as *Controller*).

Since Controller is to be primarily concerned with motion specification tasks it makes sense to implement it as an interactive system. This also fits in with the graphics environment described above. The following guidelines (Foley and Van Dam 1982) developed to enhance the user-program interaction of any graphics system are considered:

(i) keep the interaction sequences simple and consistent;

(ii) do not use too many options or different styles;

(iii) provide prompts, but ensure that they do not hamper the experienced user;

(iv) supply the user with appropriate feedback;

(v) allow the user to correct any errors that he makes.

The development of Controller's user interface is also helped by modeling it on a real life application. The production of a computer animation sequence using an interactive system can be compared to the production of a television programme. In the latter case the programme director will be in a control room overlooking the studio. From here he will coordinate the action and movement of the actors, camera crew, sound crew and stage crew. In the former case the animator will be sitting at his graphics terminal. He too has to coordinate the action and movement of cameras, cast and lights, except that the cast are computer generated, the cameras are virtual, and the lights are synthetic. As our system will mainly produce animation for the entertainment market a television control room model seems particularly apt. The analogy with a control room also elucidates why the system has been called Controller. Let us consider in more detail the activities that a television control room monitors and determine how they can be adapted to computer animation.

2.3.1 The Control Room Scenario

Careful planning and preparation take place before the shooting of a television programme commences. Scripts are written and examined by the cast and crew so that they know their role during the scene about to be filmed. The set for the scene is built and numerous lights located around it. The angle and intensity of the lights are carefully selected to give the exact lighting effect required. Cameras and microphones are also positioned and set up. Various test shots may then be made to establish camera paths and actor position and movement. The studio floor will often be crayoned or taped with toe marks and camera locations to show the final position of these paths.

The programme director takes his place in the production control room when everyone is prepared for filming. From here he can view the shot being transmitted from each camera on a series of monitors. This will enable him to select the camera shot that is to be broadcast during the filming of the scene. He will communicate with his technical crew using talk-back circuits and his floor manager will pass on any instructions to the actors. If the production is not being transmitted live he can stop the action at any point to make changes as necessary.

2.3.2 Adapting the Control Room Model

The studio control room model was chosen with the intention of creating from it a user friendly interactive graphics system. The animator becomes the programme controller and his graphics terminal is the control room. The computer thus assumes the roles of the camera and stage crews, receiving the animator's instructions from either the keyboard, or the tablet and puck. Function menus replace the communication system found in the television studio. Controller does not, however, cater for the addition of a sound track and so the role of the sound crew has so far been overlooked.

Before filming takes place a television production will be carefully rehearsed. Similarly, the scene to be animated will be carefully planned before proceeding to the animation stage. Storyboards depicting the action of the scene

(see §1.4.1) are created and used as a guideline throughout the animation process. Using information from the storyboard and elsewhere, the sets and actors required for the scene are generated (§2.4). Controller can then be used to define an animation sequence.

Controller creates an artificial television studio that uses virtual cameras, synthetic lights and computer generated actors. Methods of defining these entities within Controller are required. We have seen that one technique that can be adopted is to consider an object as a set of parameter values (§1.6.4). A camera, for example, can be parameterised by its location, orientation, and zoom angle. The location at every frame is then determined by specifying a path in some way. This technique is in keeping with the control room analogy where location marks are made on the studio floor (although the height of the object still has to be determined). Both spatial and temporal aspects of the moving object must be catered for. This is an important stage when planning animation and is discussed fully in the next chapter.

The animator still has to specify other parameter values at the frame locations defined by a path. Use of graphical valuators such as dials and sliding scales are appropriate for this task. The animator cannot be expected to define these values at every frame, however, as an animation sequence contains far too many frames. The obvious solution is to interpolate the parameter values between selected key frames (see §1.5.1). The view from each camera should also be accessible to the animator. This allows him to check the motion of a moving object and ensure that the camera is looking at the correct part of the set. If any changes are necessary they are then made before the final rendering of the sequence. The information gathered by Controller in this way is converted into three dimensional models, one for each frame of the animation sequence. Finally the frames are rendered and transferred onto film or video.

Using the above criteria the overall structure of Controller is represented in fig. 2.1 as a module hierarchy. We will now analyze the implementation of this module hierarchy.

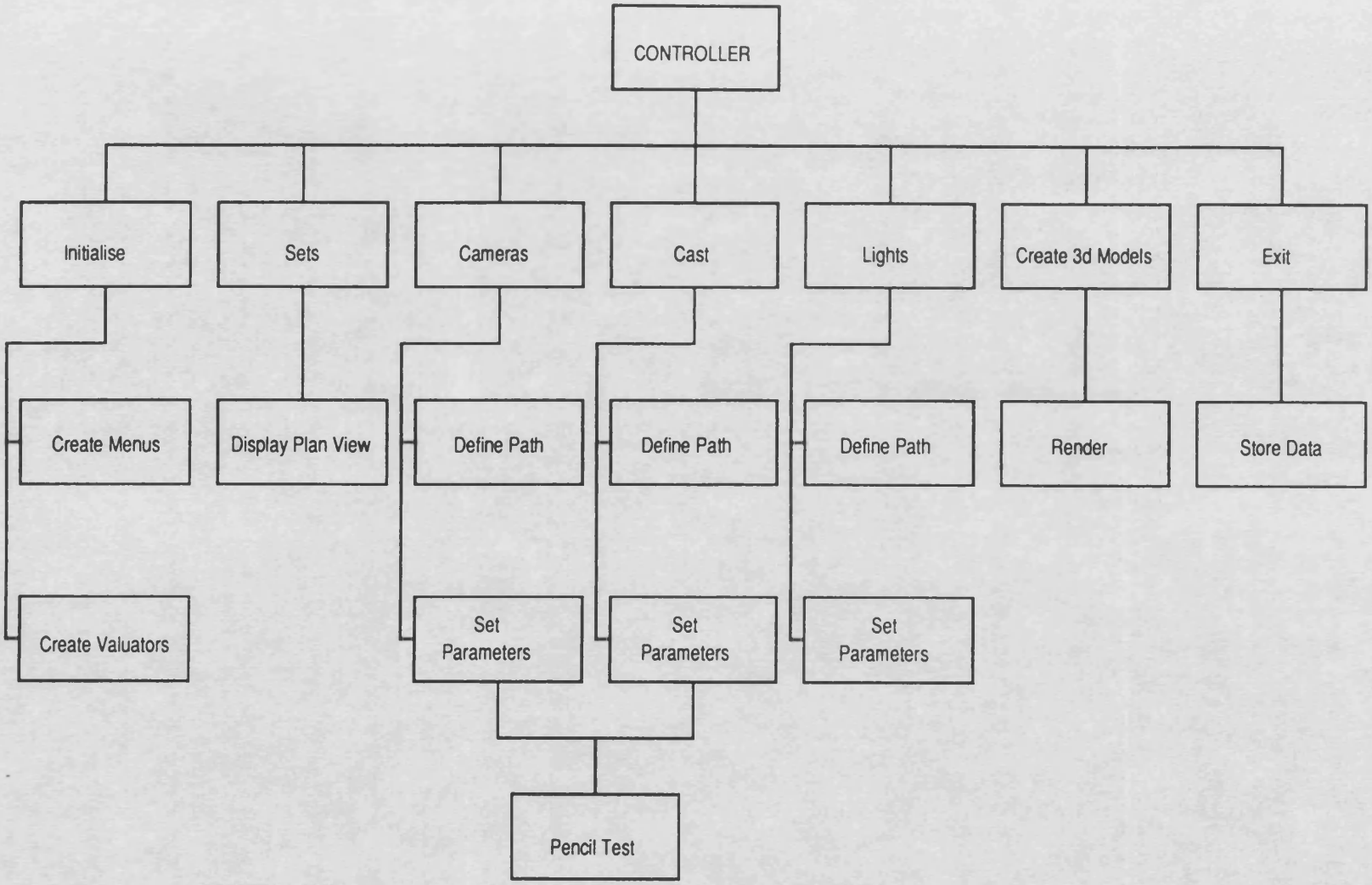


Figure 2.1. A Module Hierarchy for Controller

2.4 Sets and Cast

The first stage in computer animation production is to provide models of the actors and sets required (see §1.5.2). Many ways of building and rendering three dimensional models are available in the world of computer graphics. By providing the appropriate interface, Controller is intended to be compatible with any type of modeling system. So far, however, we have only used the in house systems described in section 2.2.5.

2.4.1 Sets

The set consists of the background and static objects about which an animation sequence is to be filmed. Before an animator can use Controller he has to create the scene model of any set he wishes to use (§2.2.5). A front elevation of this scene model will usually give a good feel for the appearance of the set and such a view is rendered using the ray tracer. When a set is required from within Controller these front elevation views are displayed on the graphics screen (fig. 2.2).

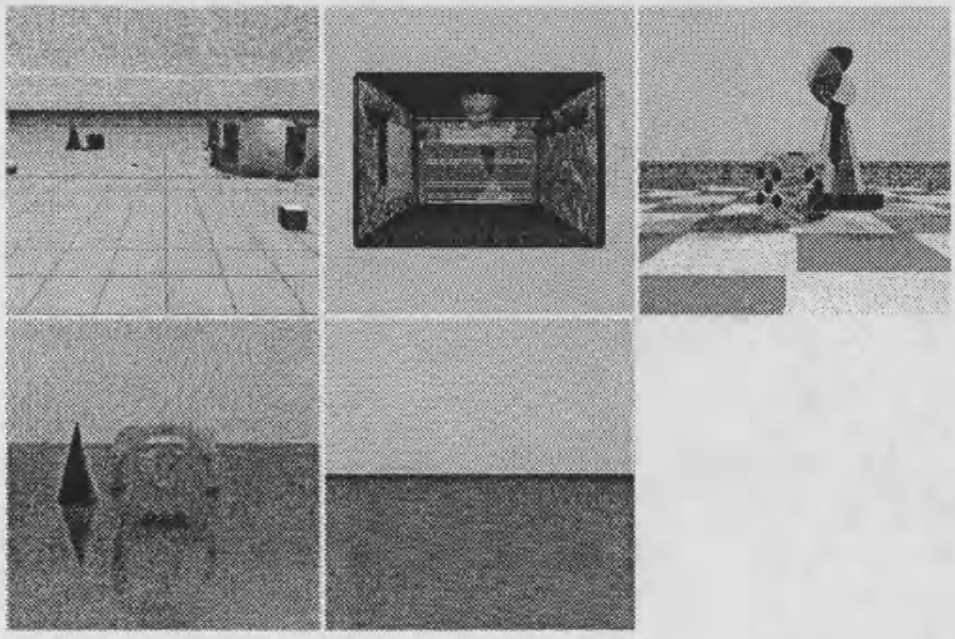


Figure 2.2. A Set Menu

The puck is then used to pick the required set from this menu. The animator also specifies the scene number and start frame number of the sequence at this point. Note that there is a maximum to the number of sets that can be displayed on the graphics screen in this way. To ensure that the animator is given the choice of sets that he requires, he specifies a file containing the titles of the sets that he is most likely to use. When invoked, Controller offers him the choice of the sets listed in this file.

Once a set has been selected its plan view is displayed at a resolution of 1024 by 1024 pixels so that it covers most of the screen. This is the same view that the programme controller will get from a window in his control room.

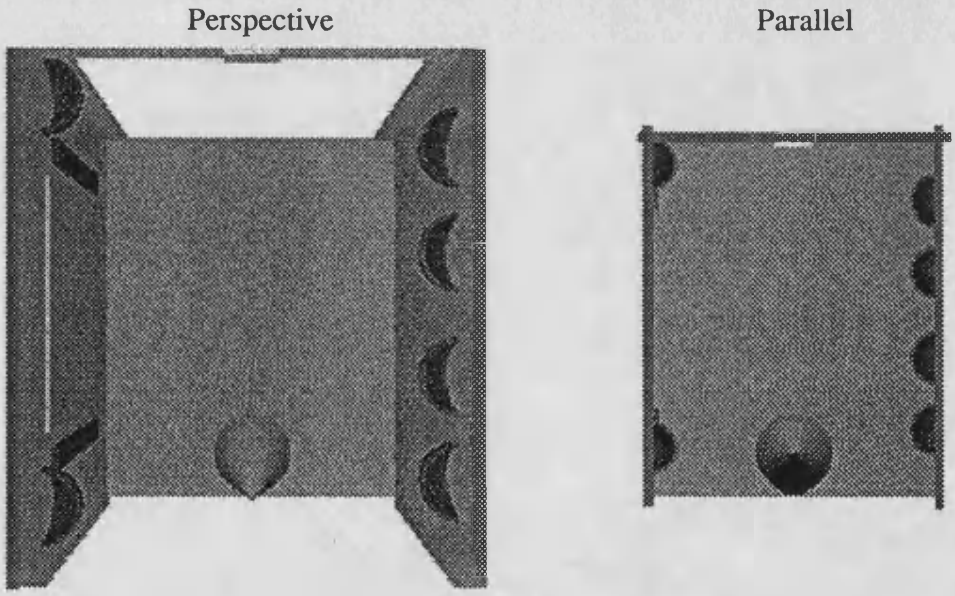


Figure 2.3. Plan View Projections

Most of the movement is planned by the animator on top of this plan view, hence the use of a large area of the screen. The ray tracer is also used here so that a realistic shaded image is obtained. Controller can, however, utilise the equivalent wire frame image if there is not enough time available for ray tracing. Note that the ray tracer can generate an image using either a perspective or a parallel projection. A perspective projection is always used for the final rendering of an

animation sequence as this is more realistic to the human eye. The animator decides on the most appropriate projection for representing the plan view of a set at the modeling stage. Figure 2.3, for example, depicts the perspective and parallel plan views of a three walled room (the second set in fig. 2.2). The view point used when generating these images is directly above the centre of the room's floor. In the perspective view the animator can clearly see the objects mounted on the walls and they do not obscure any part of the floor. The cone, however, has been distorted by the effects of perspective and may give the animator the wrong impression about its geometry. This may interfere with his design of a path. The position and orientation of the cone in the parallel view is much clearer but here the floor is obscured by the objects on the wall. This is more likely to interfere with the design of a path and so the perspective projection is preferred in this example.

When a plan view is generated using perspective, the screen on which the rendered image is formed is configured so that:

(i) its dimensions are one unit by one unit (here the screen is in the xz plane);

(ii) it is one unit away from the view point;

(iii) it is orthogonal to the view direction vector;

(iv) The view direction vector passes through the centre of the screen.

The height of the view point determines the scale of the objects in the final image. If the screen remains at a fixed distance from the view point, then increasing the height of this point above the scene centre results in a larger area of the set being captured (fig. 2.4). The objects in the set appear smaller as a result. Again the animator decides at the modeling stage what scale best suits his purposes as any path defined by him has to be calibrated to the set dimensions. A large scale is usually necessary for outside scenes while a smaller scale is enough for inside scenes. He can always make the same set available to Controller at several different scales if required. Altering the distance of the screen from the view point also affects the area of the scene covered by the final

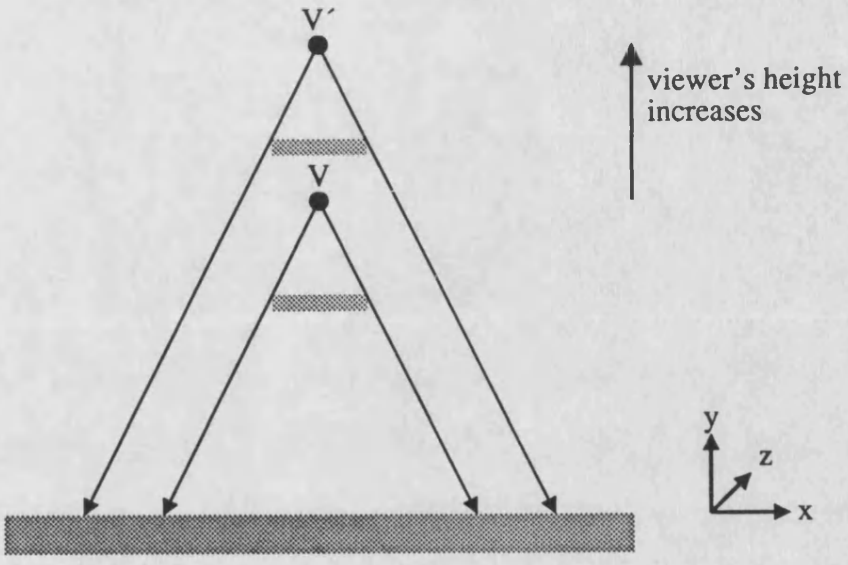


Figure 2.4. The Height of the Viewer in a Perspective Projection

image. The virtual cameras used by Controller can be made to 'zoom' in this way (see chapter 4). Note that with a parallel projection the height of the view point is immaterial. Here it is the dimensions of the screen that determine the scale of the objects in the set. In fig 2.5, for example, we require the parallel projection to use the same field of view as that used in the corresponding perspective projection. The view point in the perspective projection is at a height *h* above the scene centre. Simple geometry shows that the dimension of the screen in the xz plane should be *h by h* for the parallel projection to appear at the same scale.

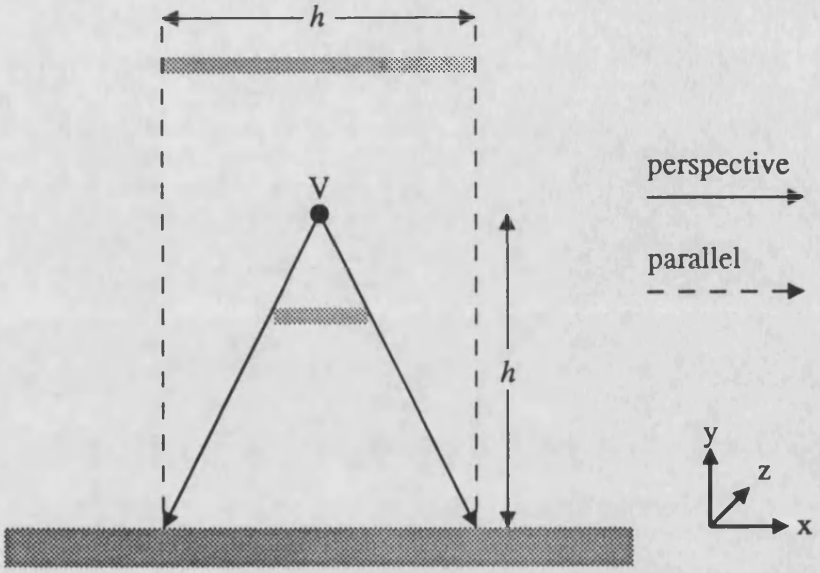


Figure 2.5. The Field of View for a Perspective and a Parallel Projection

2.4.2 Cast

A model of each new actor (or cast member) is produced in the same way as it is for a set. Appropriate views are rendered to give a clear picture of what the cast member looks like. These views are displayed on the graphics screen when the animator wants to select a cast member (fig, 2.6). Again, before activating Controller, the animator specifies a file listing the cast members used in the script. Controller allows a cast member to be used more than once in the same scene. The scale at which the cast members are to appear can also be varied by the animator. This scale is specified when using the pencil test facility and at the frame rendering stage (chapter 4).

When defining cameras and lights only their overall motion about the set is considered. A member of the cast, however, also has its own internal movement, such as limb motion. At present, Controller takes a simplified approach when providing motion of this type. For each cast member, enough poses are created to depict the motion style required. For example, the 'pacman' character in fig. 2.7 uses three poses to depict his chomp. The poses are then used in turn at

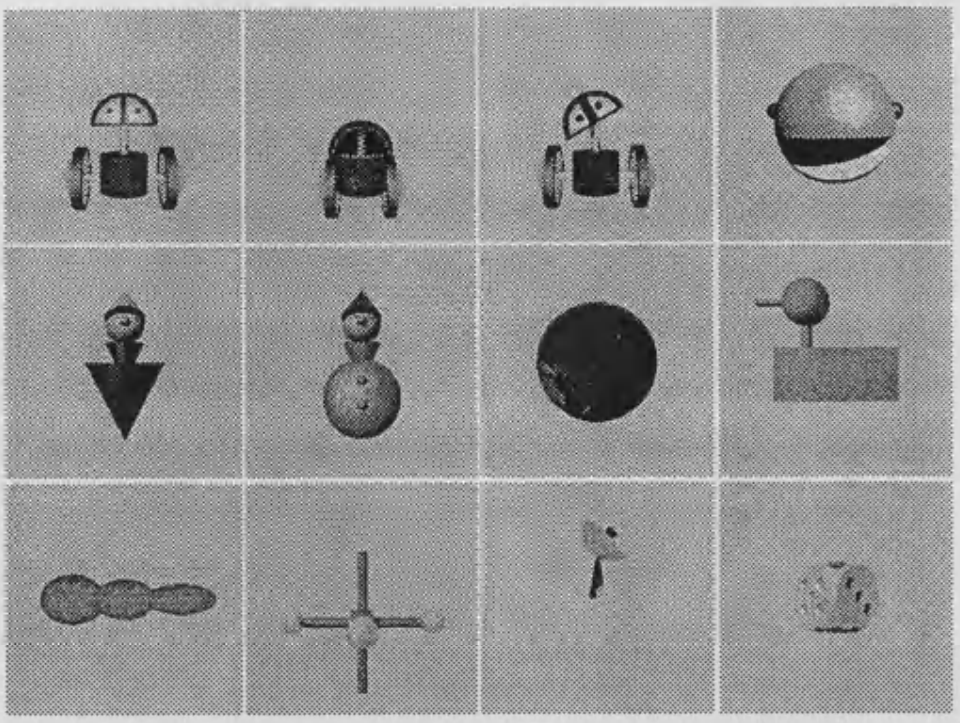


Figure 2.6. A Cast Menu

successive frame positions. If the same cast member is to have several styles of internal motion then a different set of poses depicts each motion. The animator ensures that the cast member is exhibiting the correct motion style throughout the animation sequence. Although this solution has its limitations, it is still effective in many cases (especially when a large number of poses are used). A more comprehensive solution such as the post process techniques described by Lundin (1984) will be needed in the future, however.

2.5 Designing the Implementation of Controller

The main programming language available under the UNIX environment is C and the Graphics Group's software utilities have been developed in this language. C is therefore an obvious choice for the implementation of Controller and also provides the benefits of a flexible, structured, and portable language. We will assume that the reader is familiar with the basic concepts of C as we use portions

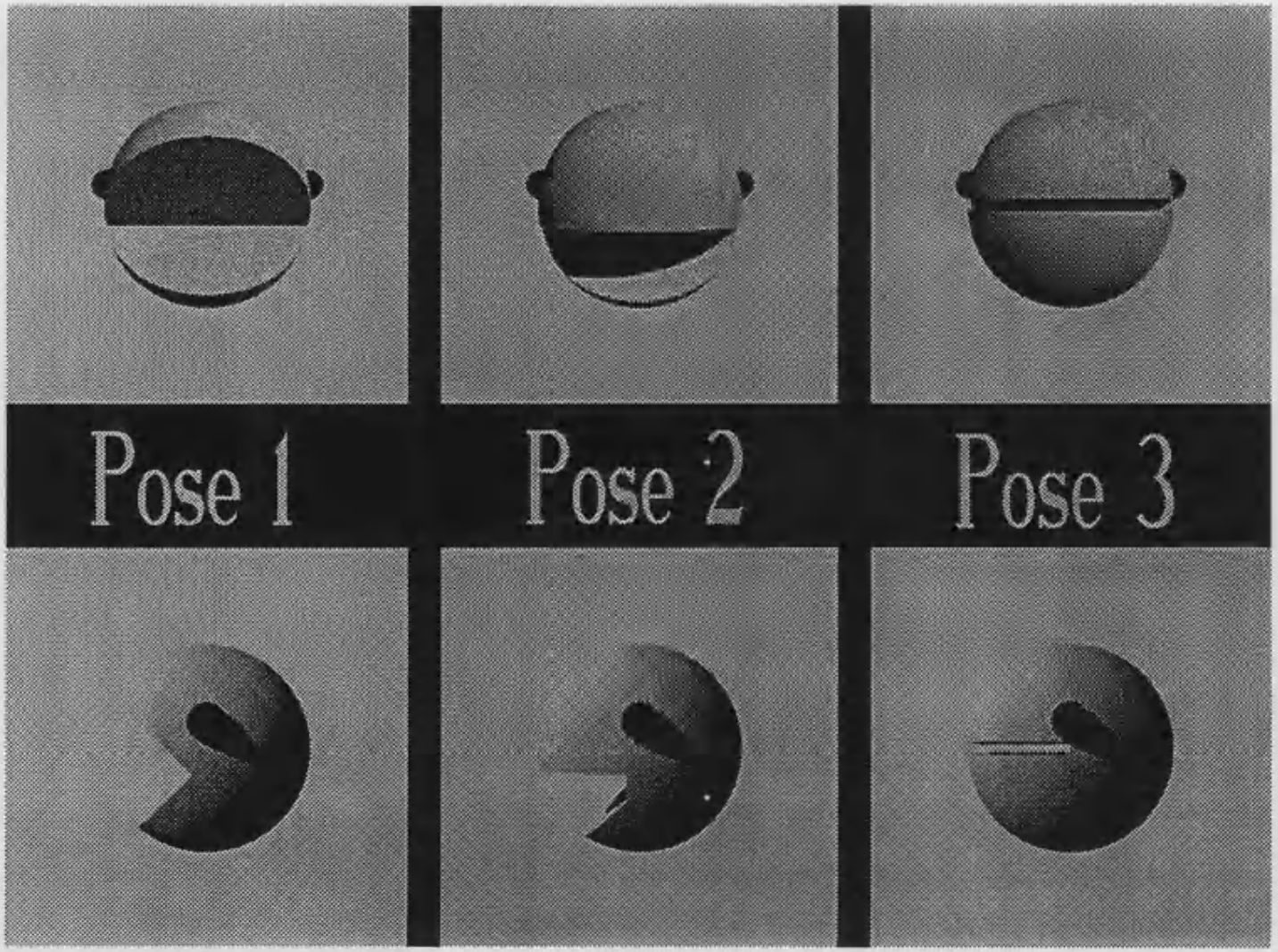


Figure 2.7. Motion Poses for a 'Pacman' Character

of code from this language to illustrate points below.

First we will apply the guidelines for good user-program interaction to Controller.

2.5.1 Interaction Sequences

The overall operation of Controller is set up as an *event driven interaction loop*, where an *event* is defined to be an action performed by the animator. The animator uses the tablet and puck as the main input device, although he will occasionally utilise the keyboard. To avoid complexity, however, these two devices are never used as part of the same function. In its default mode Controller polls the puck for its current status. The cursor on the graphics screen is then updated to reflect the position of the puck on the tablet. Whenever a puck button is pressed an *event* is triggered and the appropriate action is taken by Controller. Controller then waits for the next *event* to occur.

2.5.2 Options and Styles

Controller is driven interactively by a hierarchy of function menus displayed on the graphics screen. In keeping with our control room interface the names of these functions use terminology from television production. At the top level the animator chooses from broad categories such as camera control, lighting control, and cast control. As the menu hierarchy is descended more specific options are given. To obtain the maximum use of the screen, areas of it are not reserved specifically for these menus. Instead dynamic or pop-up menus are utilised. These are created on bit maps that can be held in memory off screen until required. They can then be copied to any position on the screen, usually at the cursor position. The area of screen being overwritten is recorded to enable it to be restored to its original state when the menu is removed.

Appropriate flags are used to keep track of where in the program hierarchy the animator currently is. They determine whether a menu should be displayed or removed and what function is being or has been selected. The *event* triggered by

the animator also depends on the current status of Controller. It may be a menu selection, keyboard or valuator data entry, or the location of some point on the screen. Always, however, an *event* occurs when data are supplied by the animator.

2.5.3 Prompts

The automatic appearance of a menu or valuator on the graphics screen prompts the animator on his next course of action. Other prompts, such as instructions to the animator, are sent to the visual display unit. Since the animator is concentrating on the graphics screen, he can ignore these instructions if he already knows what to do next. To inform the inexperienced user that an instruction has appeared on the visual display unit, however, the terminal is made to 'beep'.

Alternatively, we could supply a prompt line on the graphics screen. This would mean either reserving a specific area of the screen for the prompt line, or having a pop-up prompt line similar in use to menus. A pop-up prompt line, however, would soon become irritating to an experienced user. At the same time we did not want to reserve areas of the screen to display a prompt that the experienced animator is already expecting. For these reasons, instructions on the use of Controller only appear on the visual display unit.

2.5.4 Feedback

As Controller is a graphics system most of the feedback given to the animator is visual. For example, the menu functions selected are highlighted, valuators can be adjusted dynamically, and the path defined by the animator will change according to his instructions. Controller also uses a variety of cursors to help remind the animator about the current status (Fig. 2.8). The most important feedback for the animator comes from the *pencil test* and real time playback facilities that enable him to preview the animation. The implementation of these facilities are described in chapter 4.

Tracker: for plotting paths and selecting frame locations;

Camera: for selecting camera functions;

Pan: for specifying a camera's heading;

Zoom: for changing the lens angle of a camera;

Cast: for selecting cast functions;

Light: for selecting light functions;

Bee: to indicate that Controller is 'busy' and the animator must wait.

Figure 2.8. The Graphics Cursors used by Controller

2.5.5 Correcting Errors

Error handling facilities are built into Controller. For example, the animator is allowed to alter the shape of a path, relocate frame positions along the path, and change the value of any parameter. These facilities are covered in detail in the following chapters.

2.6 Data Structures

One consideration when implementing Controller is how to store and handle the animation data associated with an object*. We propose using sets of parameter values such as:

Cast member = {*location*, *orientation*};

* Throughout this text we use the term *object* to refer to one of a camera, a cast member, or a light.

Camera = {*location*, *orientation*, *zoom* };

Light = {*location*, *intensity* }.

These parameters can be broken down into a form in which their values can be recorded in Controller:

location: *a three dimensional coordinate, (x,y,z);*

orientation: *the way in which the object is heading, use a vector or a point in space;*

zoom: *the camera lens angle, typically between 10° and 50°;*

intensity: *the brightness of the light source, usually defined in the scale [0,1];*

Bearing in mind that even a short animation sequence is made up of many frames, efficient handling of the data is essential.

Linked lists or chains are created to store the animation data. Each node in a chain is a C structure containing appropriate fields in which to store the data for a single frame. Below, for example, is a node from a camera chain

```
struct camera_info {
     xyz_coord position;
     xyz_coord centre_of_interest;
     double lens_angle;
     struct camera_info *next_frame;
     struct camera_info *last_frame;
};
```

Note that a node contains a link to both the next and previous nodes in the chain. This is because a doubly linked chain is useful when it comes to searching the chain for data belonging to an arbitrary frame. The nodes in these *frame chains* run sequentially and so there is no need to explicitly store the frame number in each node. We just need to keep track of how far along the chain we are from the initial node. This is also the number of the frame currently being accessed.

When the animator defines a new frame location an instance of the above structure is added to the end of the chain. We are thus using dynamic memory

allocation, obtaining the storage space required at run time. Therefore, the only limitation on the length of a frame chain is the amount of memory available on the host machine. This is far more satisfactory than using an array to store the frame data. The length of an array has to be determined before the source code is compiled but at this stage we do not know what this length should be.

In all, three types of chain are set up, one each to deal with cameras, cast and lights. Every object has its own chain and there are usually several of them in existence at one time. A link to the initial node of each chain is maintained to allow us access to and to switch between them as required. The links held within a node can then be used to move along the chain so that data are stored or retrieved as desired.

A node is defined by one of three different parameter sets and so its size depends on the object it belongs to. Storage fields such as the frame location coordinate are common to all three nodes, however. It would be efficient if the same source code is used to access one of these common fields without having to determine on each access the type of chain being used. The node cannot normally be passed in the argument list of some general procedure, however. We do not know in advance which of the three data types the node argument has to be declared as within this procedure. The solution adopted is to use pointers to functions. Consider the problem of extracting the frame location coordinate from an arbitrary node. Three simple functions are provided to return this coordinate from each of the node types. For example, the camera function is

```
int get_camera_coordinate(x,y,z)
double *x, *y, *z;
{
     *x = camera_link->position.x;
     *y = camera_link->position.y;
     *z = camera_link->position.z;
};
```

A global pointer is set to this function whenever the animator selects a new camera, that is

```
get_coordinate = get_camera_coordinate;
```

Similarly for the cast and lights. A common procedure can then be written to access the required coordinate using this function pointer as follows

```
(*get_coordinate)(&x,&y,&z);
```

Once the global function pointer has been set there is no need to determine the type of chain being used at each access. This technique is also applied to storing the object position and orientation at one of its frames.

2.7 An Object Orientated Environment

Controller has been implemented in an *event driven environment* (§2.5) using general purpose procedures (§2.6). An alternative approach would be to include procedural information as part of a property list belonging to some specific object. New facilities can then be included by extending this property list as appropriate. An environment that is set up in this way is often referred to as an *object orientated environment.* Procedures called *methods* can be attached to an object and executed whenever an appropriate message is sent to that object.

As we have already described, the director of a television studio will coordinate the production of a programme by sending messages to the stage and camera crews. This scenario is an ideal candidate for being modelled in an object orientated environment. We therefore decided to produce an experimental version of Controller using:

(i) the C++ programming language, and

(ii) UNIX message sending facilities.

Our intention is to use these to simulate an object orientated environment.

2.7.1 Experiments with C++

C++ is a development of the C programming language and retains C as a subset. Its compiler acts as a preprocesser for the normal C compiler. The main ways in which C++ differs from C are by the provision of:

(i) type checking of function arguments;

(ii) modular design;

(iii) classes and inheritance;

(iv) operator overloading.

Full details of these and other differences are given by Stroustrup (1986).

C++ offers features found in an object orientated environment. In particular, it allows types called **classes** to be defined for which access to data is restricted to a specific set of functions (the member functions). An object is declared to be a member of a certain class in the same way as variables are declared to be, for example, of type integer. A hierarchy of classes can then be created by deriving further classes from the base class. An important feature of such a hierarchy is that a derived class **inherits** the properties of its base class, in addition to any new properties defined just for it.

A subset of Controller was converted into C++ so that a class structure could be incorporated. This version of Controller allows for path specification and the plotting of frame positions along the path. It also contains the code for the selection of the frame positions between which some parameter is to be defined. Detailed descriptions of these facilities are not required here as they are covered in the following chapters.

When a path is drawn onto the graphics screen, a container class is required to store the path coordinates, for example

```
class path {
     int sz;
     int* x;
     int* z;
public:
     path(int);
     void store(int,int,int);
     void get(int,int*,int*);
};
```

The variables `sz`, `x` and `z` are only accessible to the member functions **path, store** and **get.** The function **path** is called the constructor function and initialises `x` and `z` to point to vectors of length `sz`. These hold the coordinates that are accessed by the **store** and **get** functions. A container class is also required to hold the cast or camera frame positions along such a path

```
class movable {
     friend class camera;
     friend class cast;
     int *x;
     int *y;
     int *z;
     int sz;
     int choose_from_range(int);
public:
     movable();
     int storex(int,int);
     int storey(int,int);
     int storez(int,int);
     int get(int,int*,int*,int*);
     int get_pos();
};
```

Any object declared to be a member of this class is automatically provided with functions to access its frame location coordinates. A subclass can then be derived from **movable** to deal with options related specifically to cameras, for example

```
class camera: public movable {
     int* zf;
     int* ix;
     int* iz;
public:
     camera();
     void set_heading(int,int,int);
     void get_heading(int,int*,int*);
     int set_zoom(int,int);
};
```

The specification of a cameras heading and zoom are thus provided for. A class dealing with cast specific functions can similarly be derived. Note that the base class **movable** has to declare both **cast** and **camera** as being a **friend.** This enables the derived classes to access the variables that would otherwise be private to **movable.** The derived class **camera** inherits the properties of the base class as well as having its own specific properties.

2.7.2 Sending Messages

An important feature of an object orientated environment not supplied by C++ is the ability for objects to communicate with each other by sending messages. An attempt was made to simulate message sending by using the UNIX toolkit of **forks, system** calls, **pipes** and **sockets.**

A **fork** creates a new (child) process. The child process is an exact copy of the parent process except that it has a unique process identifier.

A **system** call can be used to execute a process from within a different process. Control is then returned to the calling process.

A **pipe** can be used to send data from a parent process to a child, or vice versa:

Parent:

Make a connection to the child:

```
fout = popen("child","r");
fprintf(fout,"Information from the parent");
```

Close the connection:

```
pclose(fout);
```

Child:

Read from parent and output the message:

```
while( (c=getchar()) != '0 ) putchar(c);
```

However, when two way communication between processes is required it is simpler to use **sockets**, in particular a **socketpair** which creates a pair of connected sockets. Figure 2.9 provides a simple example where data are sent to and from the parent and child.

If two or more processes require use of the graphics screen then our task is made more complex. Normally a screen lock ensures that only one process is allowed access to the graphics screen at any given time. We can override the screen lock, but there are still problems. Before a process can use the graphics screen information internal to the graphics system is initialised. This information must be available to all processes using the screen or it is still impossible for more than one process to have access to it. This rules out the use of a **system** call to initialise animation functions under the control of other processes. To ensure that the required information is available to each process, the entire source code has to be copied by using a **fork**. Each time a process is spawned in this way, however, there is a substantial (and undisirable) increase in the memory used by the system.

2.7.3 Comments

There is no noticeable difference in the performance of either the C or the C++ version of Controller as far as the animator is concerned. The C++ version, however, produces larger executable code owing to the extra overhead involved. So what are the advantages, if any, of using the C++ programming environment?

```
main()
{
      int sv[2]; - contains the two socket descriptors
      char buf1[20], buf2[20];

      Make a two way connection:
      socketpair(AF_UNIX,SOCK_STREAM,0,sv);

      Use a fork() to create a child process:
      if(fork() == 0)
      { In the child

            Receive a message from the parent:
            read(sv[1],buf2,sizeof(buf2));
            shutdown(sv[1],0);

            Send a message to the parent:
            buf1="Message_from_child";
            write(sv[1],buf1,sizeof(buf1));
            shutdown(sv[1],1);
            exit(1);
      }

      else
      { In the parent

            Send a message to the child:
            buf1= "Message_from_parent";
            write(sv[0],buf1,sizeof(buf1));
            shutdown(sv[0],1);

            Receive a message from the child:
            read(sv[0],buf2,sizeof(buf2));
            shutdown(sv[0],0);
      }
}
```

Figure 2.9. Communicating Between Two Processes Using a Socketpair

The inheritance feature encourages more economical code to be written and it is a lot easier to share code in this way than having to manipulate pointers to functions. The class structure with its 'private' variables helps to locate code and make debugging easier. The strict type checking performed by C++ also helps to

reduce bugs. Despite these features, it did not seem to be worthwhile converting the whole of Controller into C++. A useful facility of an object orientated environment is the ability of objects to communicate with each other by message passing and this is not available in C++. Although we attempted to overcome this deficiency, our experiment was not a great success. We therefore decided to continue the development of Controller using our event driven approach.

2.8 Summary

This chapter has introduced the Controller system of animation planning. After describing methods of modeling actors and backgrounds, we considered the design of Controller's user interface. By basing this on the operation of a television control room we have provided a foundation for an easy to use interactive graphics system. We have also examined different approaches for coding the system and found that an event driven approach is suitable for our requirements.

We will now look in more detail at techniques that can be used to carry out the animation planning.

Chapter 3

Motion Paths

3.1 Introduction

The main function of Controller is to allow the animator to plan the movement of cameras, cast members, and lights. He achieves this task by:

(i) defining a *motion path* for each object;

(ii) setting the parameters associated with an object at each frame.

This chapter describes the first of these stages (see also John and Willis 1989b).

A *motion path* will determine the overall position of an object during the scene being animated. To specify such a path, the animator will have to supply Controller with both spatial and temporal information about the object. In both cases we aim to provide a flexible interface offering him fine control over the motion specification. The chapter begins by following the procedure that the animator uses to provide the spatial information. We then present and compare various methods by which the temporal definition can be implemented. Our main objective is to achieve smooth motion but we also examine whether the mass of an object can be simulated.

3.2 The Spatial Definition

At this point the animator has already chosen the set to be used for the scene and its plan view is on the graphics screen (see §2.4.1). He will now choose the cast member, camera or light that he wishes to define a motion path for, and will commence its spatial definition. In general we have to define a three dimensional path using a two dimensional device so we consider first only the xz or ground plane. The extension into three dimensions by the setting of the object's height

parameter will be described in the next chapter. The animator will therefore proceed to draw a two dimensional *track* onto the plan view of the set. The object will move along this track during the period that it is present in the scene.

Controller provides the animator with several methods of using the tablet and puck for drawing tracks. One option allows the animator to define a track using a series of straight line segments. If selected, a line is 'rubber banded' from the part of the track already drawn (if any) to the current cursor position. Pressing a puck button will fix the line segment, and so on. Although a track defined in this way is sometimes appropriate (a camera often tracks along a straight line), natural movement tends to follow arcs rather than straight lines. A circular track can be drawn by the animator but although this facility is useful, motion in a circle can often look too mechanical. Another option is to allow the animator to draw a track freehand. However, it is difficult to draw the track smoothly in this way and the resulting motion will often appear erratic. A method of representing natural movement in a more convenient way was therefore sought.

3.2.1 Spline Curve Formulations

If a draughtsman wishes to produce a smooth curve he will use a flexible strip called a spline to draw through a set of control points. Spline curves can be represented mathematically and their application to computer graphics is well known. We decided to include such spline techniques in Controller so that the animator can construct a track out of smooth curve segments. Several different forms of splines exist (see Foley and Van Dam 1982), but they are all based on the parametric representation of a three dimensional curve

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \ ; \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \ ; \\ z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z \ . \end{aligned} \qquad (3.2a)$$

We are dealing with finite curve segments so t is usually limited to the range [0,1]. A spline curve is made up of a series of these curve segments linked

Catmull-Rom spline is the Hermite spline where the tangent vector, D_i is calculated as $D_i = \frac{1}{2}(P_{i+1} - P_{i-1}) = \frac{1}{2}((P_{i+1} - P_i) + (P_i - P_{i-1}))$

i.e. its the average of the two adjacent cords between the control pts.

together so that there is continuity of position and slope across the joins. A true spline has second order continuity (for example, B splines and β splines), although some spline curve formulations provide only first order continuity (for example, Hermite and Bezier splines).

The Hermite formulation produces a curve with first order continuity. Each curve segment is defined by two control points and the tangent vector at these points. The curve produced will pass through the control points.

Bezier curves also have first order continuity. Here the curve segments are defined by four points and tangent vectors are not needed. A Bezier curve is thus easier to define than a Hermite curve. Two of the defined control points are the end points of the curve segment and all four control points form a convex hull in which the curve segment will lie. This ensures that the curve will smoothly follow its control points without erratic oscillations.

The B spline is an approximating spline and so does not pass through its defining control points. Its formulation, however, produces a curve that has second order continuity. This means that each curve segment making up the spline is joined with the next in such a way that the first and second derivatives across the join are continuous. A much smoother curve is therefore produced. The B spline also has the convex hull property.

Second order continuity is also provided by β splines. Two parameters (called beta parameters) are introduced to provide greater manipulation of the shape of the spline. The beta parameters adjust the shape of the curve segments about the convex hull of their control points.

We chose to use B splines in Controller. The smooth curve produced by splines with second order continuity are more suitable for plotting tracks that can represent natural motion. They will prevent discontinuities in the direction of motion from occurring. We do not need the extra shape manipulation provided by β splines, however, the B spline will be satisfactory. The formulation given by Foley and Van Dam (1982) is of the form

$$x(t) = TMG_x,$$

where

$$T = [\, t^3 \; t^2 \; t \; 1 \,], \quad M = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}, \quad G_x = \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix},$$

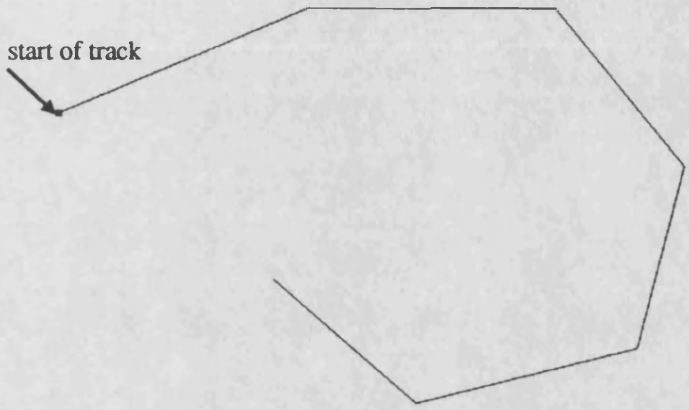
$P_0 \cdots P_n$ are the control vertices.

The use of a cubic spline means that a track could be defined through the set in three dimensions. However, providing an easy to use interface for the animator to do this is not straightforward. We therefore restrict the spline to two dimensions when carrying out the track drawing stage. This also simplifies the calculations involved. As we have already stated, the setting of an object's height will be discussed in the next chapter.

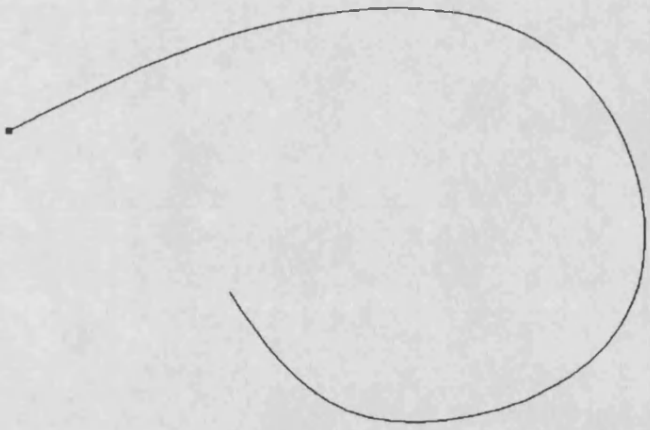
3.2.2 B Spline Tracks

To define a B spline track in Controller the animator begins by drawing a series of (rubber-banded) straight line segments onto the plan view of the set (fig. 3.1). The end points of these segments are the points to be interpolated by the spline and they approximate the desired shape of the track. A B spline does not generally interpolate any of the control points defining it, however, it just passes close to them. If the end points of the straight line segments are used as control vertices then they will not lie on the track as desired. We therefore have to calculate the control vertices that will result in the interpolation of the desired points. To do this Controller uses a method described by Barsky and Greenberg (1980) but we only have to apply their method to two dimensions. It works by deriving and solving a set of linear equations that express the interpolation condition for each point about the unknown control vertices. Specialised algorithms for evaluating these unknowns are also provided.

Controller composes the shape of each spline segment by using ten coordinates joined by straight lines. These are usually enough to draw a visually smooth curve although the resolution of the segment can be altered as desired. The animator also has the option of defining the B spline curve to form a closed



a. Defining the shape of the track



b. The interpolated B spline

Figure 3.1. Defining a B Spline Track

loop. Below, for example, is the track obtained by interpolating a closed B spline through the same defining points as those used for the open spline in fig. 3.1.

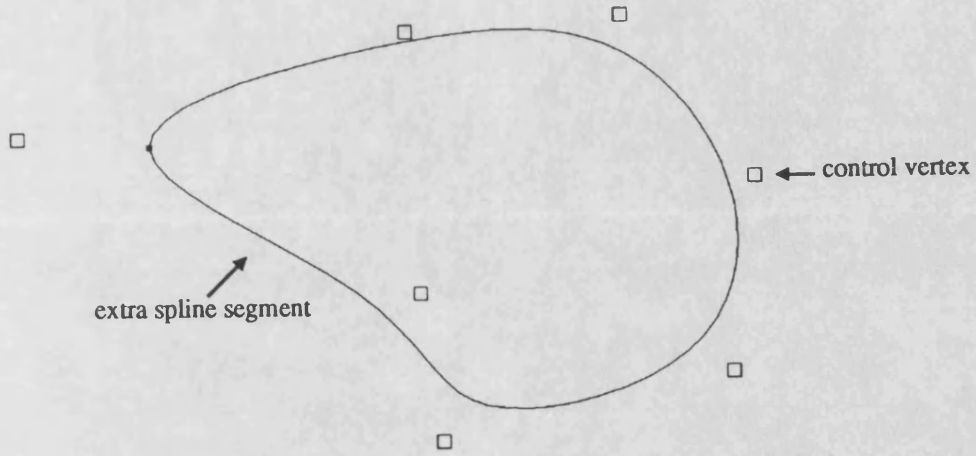


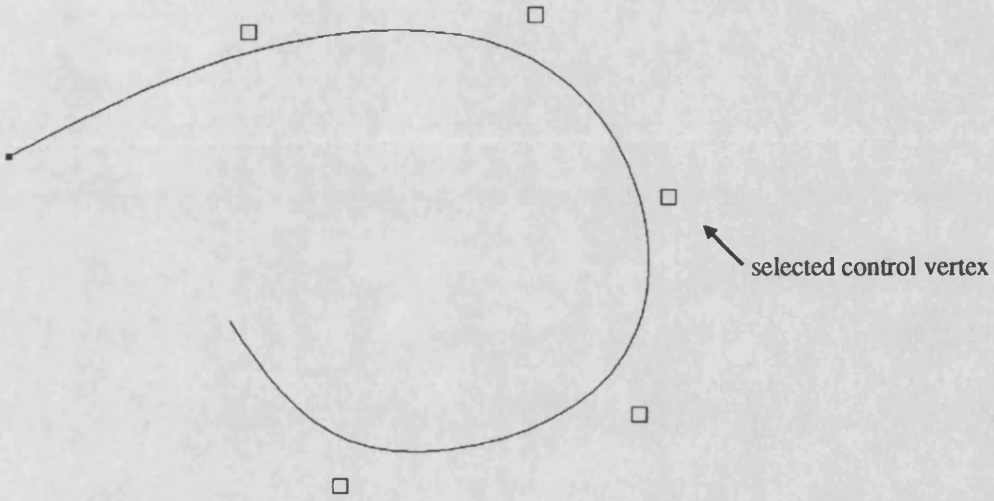
Figure 3.2. A Closed B spline

If a B spline is closed in this way then an extra cubic segment between the start and end points has to be drawn. The control vertices calculated at the end points of the track differ for the open and closed splines.

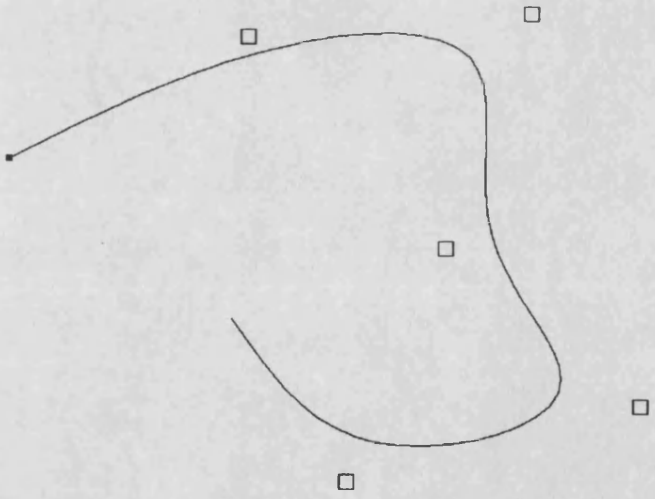
3.2.3 The B Spline Editor

An important part of the design of Controller are the provision of error correcting facilities. An obvious place for such a facility is at the track drawing stage, especially when B splines are being used. The animator is often unhappy with the B spline track after it has been interpolated as it may, for example, be too close to some object in the set. Controller provides a B spline editor to allow the track to be adjusted accordingly.

When the B spline track is drawn, the position of the control vertices that define it are also displayed on the screen (fig. 3.3). The animator can use the puck to select a control vertex and move it to a new position. The B spline track is then redrawn to reflect this change. An advantage of the B spline formulation



a. The original track



b. The updated track

Figure 3.3. Editing a B Spline Track

is that each control vertex only exerts a local influence on the spline. Therefore, the whole spline curve does not have to be redrawn as only the two cubic segments influenced by the selected control vertex have to be erased and recalculated. This process is illustrated in fig. 3.3.

To erase a section of track the screen pixels that define it must be restored to the original colour of the set. If the track is plotted by complementing the colour of each pixel that it covers then this is a simple task. By complementing the colour of the pixel a second time it is restored to its original state. Because of the dithering techniques used by the ray tracer when generating images, however, this method is unsatisfactory. Areas in the set picture that look as if they are of constant colour are made up of many different coloured pixels. When lines of complemented pixels are drawn on top of such an area they appear to change colour frequently and are often difficult to discern.

Visibility is not a problem if tracks are plotted using lines of pure colour. Such tracks show up clearly on the set and can also be colour coded to help the animator identify the type of object that the path represents*. Therefore, when a track is plotted the colour values of the pixels being overwritten are placed onto a stack. To erase the track it is plotted again, but this time in reverse. Each pixel is assigned a colour value popped off the stack and is thus restored to its original colour. Note that one byte of storage space is needed for each pixel value recorded on the stack. Extra storage space is therefore required when using this method but the improved visibility of the tracks make this cost worthwhile.

3.3 The Temporal Definition

So far we have been generating positional information without any reference to time. Therefore, when the animator is satisfied with his spatial definition the next step is to decide where the object will be at each frame. Remember that an animation sequence is displayed at a constant rate and so Controller has to

* Controller uses red for cameras, green for cast members, and blue for lights.

calculate the position of the object at fixed time intervals. The method used to achieve this will be an important factor in determining the effectiveness of the final motion. In Controller we do not use splines and complicated physical laws for this purpose. The latter produce realistic results but are also computationally expensive. Further, both techniques increase the difficulty the animator has in specifying the exact motion he requires. We define the temporal information of a motion path using kinematic techniques as these provide the animator with a more intuitive method of carrying out this task. To produce realistic results we rely on the animator's skill and the flexibility of the system.

An object in motion will be accelerating, decelerating, or moving at a constant speed. First we describe how the animator specifies the temporal information to simulate these styles of motion.

3.3.1 Motion Segments

The direction in which an object traverses its track is in the same direction as that in which the track is drawn. The resulting motion is described by a series of *motion segments*. Typically, the animator places three constraints on each motion segment he defines:

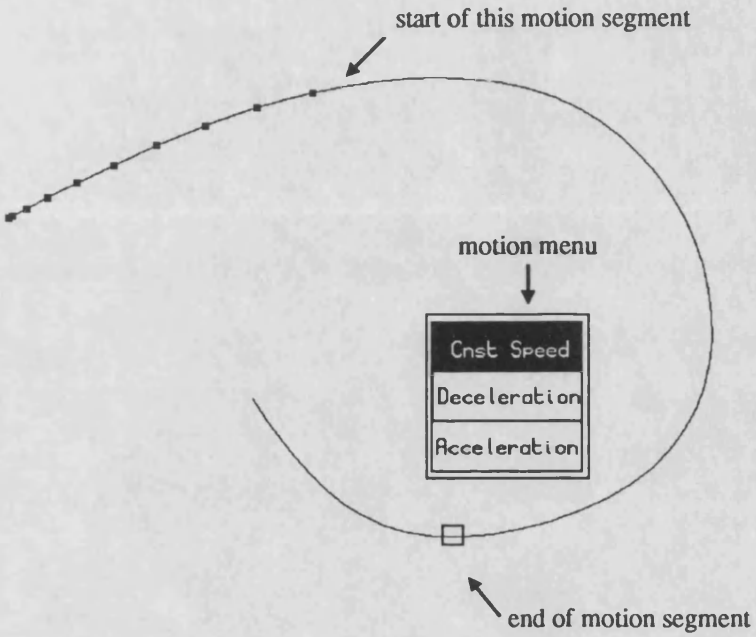
(i) the length of track to be used;

(ii) the number of frames to be taken;

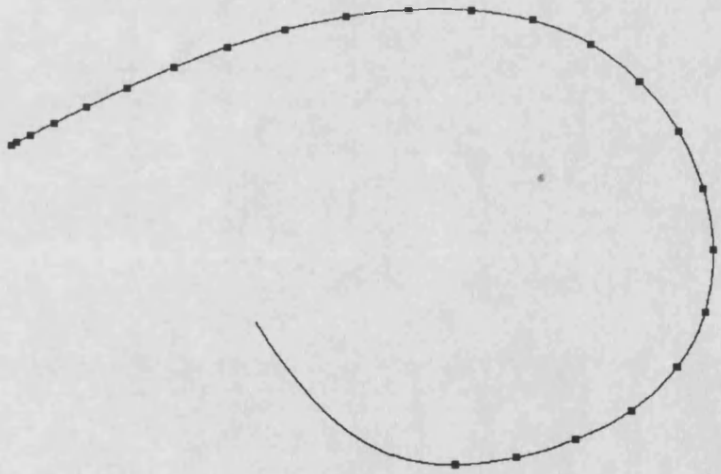
(iii) the motion style (acceleration, deceleration or constant speed).

The length of track used is taken from the most recently calculated frame position (the beginning of the track or the end of the previous motion segment) to some point that the animator indicates on the path by using the puck (fig. 3.4). This point is identified by searching through the track coordinates for the nearest one to the puck when a puck button is pressed.

The animator then selects the motion style required from the menu displayed. If acceleration or deceleration is selected then the animator is also required to enter the duration (in frames) of the motion segment. The procedure is slightly different if the object is to move at a constant speed. Here the animator only



a. The current motion path



b. The updated motion path

Figure 3.4. Defining a Motion Segment

indicates the distance to be covered during the motion segment and as many frames as possible are then fitted in. The frame locations that Controller calculates to satisfy the animator's specification are indicated on the track by plotting small squares. In fig. 3.4 we can see from these location indicators that the first motion segment defined is an acceleration phase lasting for nine frames. The animator then specifies a constant speed motion segment. Note that he can also make the object remain at the most recently calculated frame position for any number of frames. Before the methods used to calculate the distance travelled between frames are discussed, we will detail how the location indicators are plotted onto the track.

All tracks (see §3.2) are constructed out of straight line segments, even curved segments of a track are made up of several small straight lines. The coordinates of the end points of each line segment are known and so it is straightforward to calculate the length of these segments. This procedure is computationally fast to carry out and gives a good approximation of the length of the track. Let us assume that the motion modeller calculates that the next frame location is at a distance, d, from the previous location. The particular line segment along which this location lies is determined by first subtracting from d all successive line segments that can be completely engulfed. This will leave a residue, r, that is the distance along the identified segment of the required frame location. In the example of fig. 3.5, a constant speed motion segment is being evaluated. Here

$$\vec{p} = r\vec{QR}$$

where $\vec{p}$ is a vector denoting the required frame position, and so

$$p_x = Q_x + r \sin\theta,$$

$$p_z = Q_z - r \cos\theta.$$

The angle θ is obtained using simple geometry. The calculation of the values of p_x and p_z depend on the direction of the line segment vector. Here x is

--

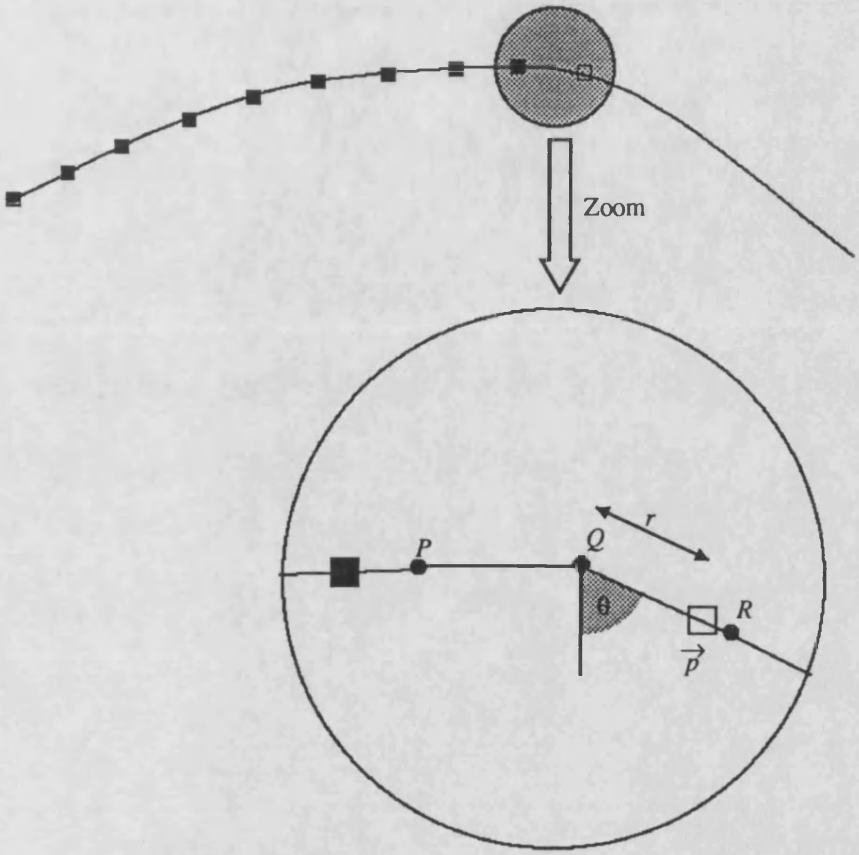


Figure 3.5. Calculating the Frame Locations

increasing and z is decreasing. Note also that the distance, $QR - r$, must be taken into account when calculating the next frame position.

3.3.2 The Undo Command

Motion segments can be combined in any manner and so many motion effects can be achieved. The animator decides whether the resulting motion appears realistic or unrealistic. Any motion segment that he is not satisfied with can be undone and another attempt made. The values of the pixels overwritten by the frame location indicators are recorded in the same manner as when a track is drawn. These pixels are restored when a motion segment is undone so that the squares are erased. All motion segments that have been defined can be removed in this

way. The whole path, that is the spatial and the temporal definition, can be completely removed if desired.

3.4 Achieving Smooth Motion

This section presents several kinematic methods that can be used to calculate motion segments. These methods calculate positions, speeds and accelerations as a function of time. We also have to consider the interchange between successive motion segments. Usually we will require this interchange to occur smoothly so that it cannot be detected by the viewer. Note, however, that the animator will not always require smooth flowing motion and so he should be able to prevent this from occurring.

3.4.1 Trigonometric Functions

In key frame interpolation, acceleration and deceleration effects are often modelled using

$$y = 1 - \cos(x), \quad 0 \le x \le \pi; \tag{3.4a}$$

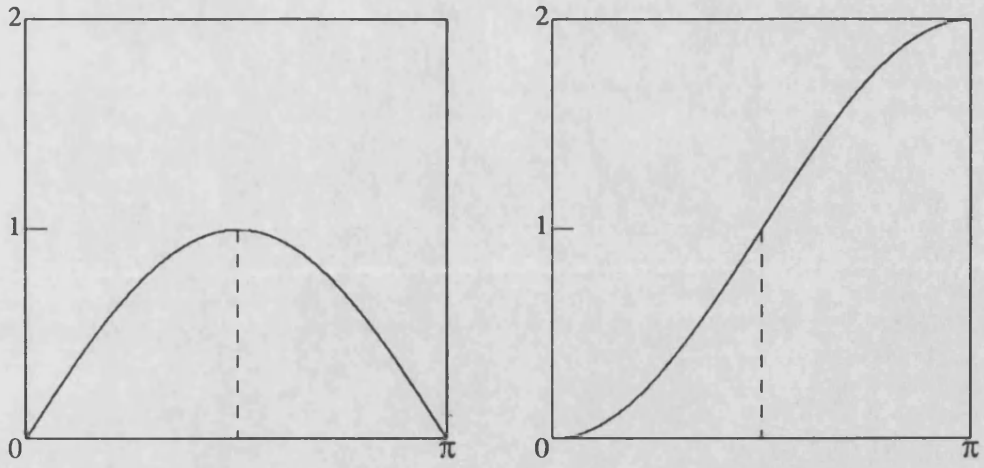
$$y = \sin(x), \quad 0 \le x \le \pi/2 \tag{3.4b}$$

respectively (Magnenat-Thalmann and Thalmann 1985). We can see why such motion effects are obtained from these functions by considering their graphs as functions of distance against time (fig. 3.6).

Suppose that the animator wishes to define a motion segment for an accelerating object. He supplies the length of the track to be used for this motion segment, call this L_s, plus its duration in frames. Applying (3.4a) over the range $[0,\pi/2]$ gives the required acceleration effect. We calculate the distance along the track at each frame of the motion segment using

$$l(t) = L_s \times (1-\cos(\pi/2 \times t)), \tag{3.4c}$$

where $l(t)$ is the proportion of L_s traversed up to time t. The value of t must fall into the range [0,1] to be used with (3.4c) and so it is scaled using



$s = sin(t)$ $\qquad$ $s = 1\text{-}cos(t)$

Figure 3.6. Motion Curves for the Trigonometric Functions

$$t = \frac{current\ frame\ of\ this\ motion\ segment}{duration\ in\ frames\ of\ this\ motion\ segment}\ .$$

If the animator requires the motion segment to show acceleration followed by deceleration then (3.4a) is applied over the range $[0,\pi]$. The motion segment equation is now

$$l(t) = L_s \times \frac{1-\cos(\pi \times t\ )}{2}\ . \qquad (3.4d)$$

Similarly, to obtain just a deceleration effect, (3.4b) is applied to the motion segment producing

$$l(t) = L_s \times \sin(\pi/2 \times t). \qquad (3.4e)$$

The animator may require an object to continue at a constant speed as well as accelerating and decelerating it. This speed will be that attained by the object at the end of the motion segment last calculated. A reasonable approximation of this is to take the average speed of the object between the two most recently

calculated frame positions. This is, in effect, the distance between these two frame positions.

The overall motion definition consists of some combination of motion segments. Suppose, for example, that the animator defines a sequence of motion segments when an object:

(a) accelerates from rest;

(b) maintains a constant speed;

(c) accelerates again;

(d) decelerates.

The graph of distance against time for this motion definition is given in fig. 3.7. We require a smooth interchange between each motion segment. Note that if two or more successive motion segments are of the same style, acceleration for example, then the best results are obtained by combining them into one motion segment. We thus have less interchange points to consider.

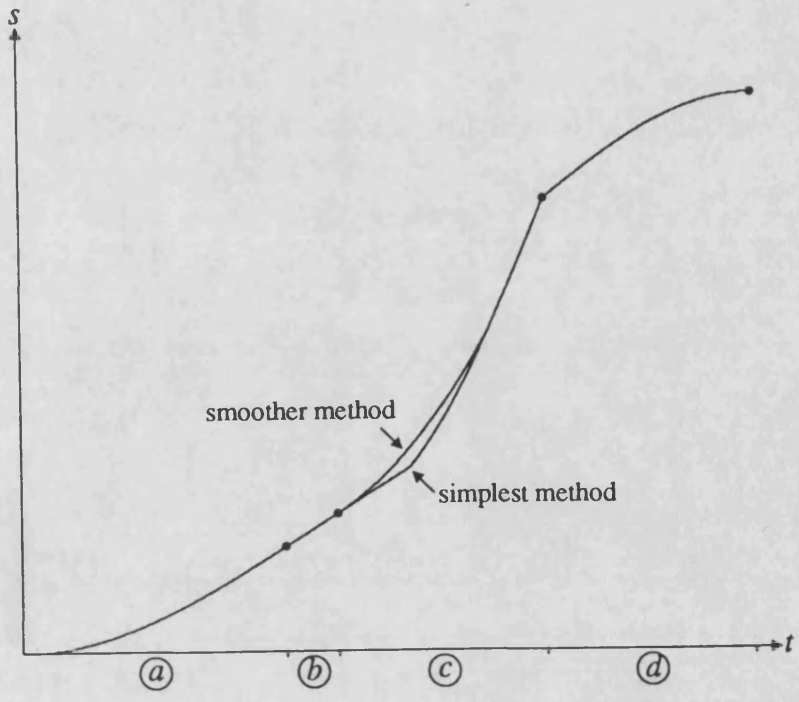


Figure 3.7. Using the Trigonometric Functions

The first motion segment of our example presents no problem as we can satisfy the distance and time constraints using (3.4c). The third motion segment also involves acceleration, but here we have to take the current speed of the object into account. The acceleration function does not do this and so cannot be used as it stands. We adopt the following simple solution to this problem.

The distance yielded by the acceleration function is only used if it is greater than the distance that would be covered by the object continuing to move at its current speed. Until this happens the object's speed is not altered. The point at which the acceleration function takes over can be noticeable, as it is in fig. 3.7. Further, if we increment the object's speed after every frame by some appropriate amount so that the object does indeed appear to accelerate, then the change over point is much smoother. Up to the change over point the object's speed increases in an arithmetic progression. A similar technique can also be used when modelling deceleration using (3.4e). Here we have to ensure that the speed of the object is always less than its speed coming into the current motion segment. We have not needed to do this in the above example as the initial speed obtained from (3.4e) is a lot less than the speed of the object at the end of the third motion segment. Unfortunately the difference is too great for a smooth interchange.

Sometimes the constraints that the animator defines make it impossible (perhaps deliberately so) for the desired smooth interchange between successive motion segments to be achieved. We can check for such cases and warn the animator who may then decide to change his motion definition.

3.4.2 Sinusoidal Fairing

The process of accelerating an object from rest to some steady speed is sometimes called in-fairing. A technique known as *sinusoidal fairing* offers an alternative solution to providing a smooth interchange between the two motion styles involved here (Kingslake 1986). An object decelerating to rest from a constant speed (out-fairing) can also be taken care of.

In fig. 3.8, arcs AB and CD are quarter circles, so

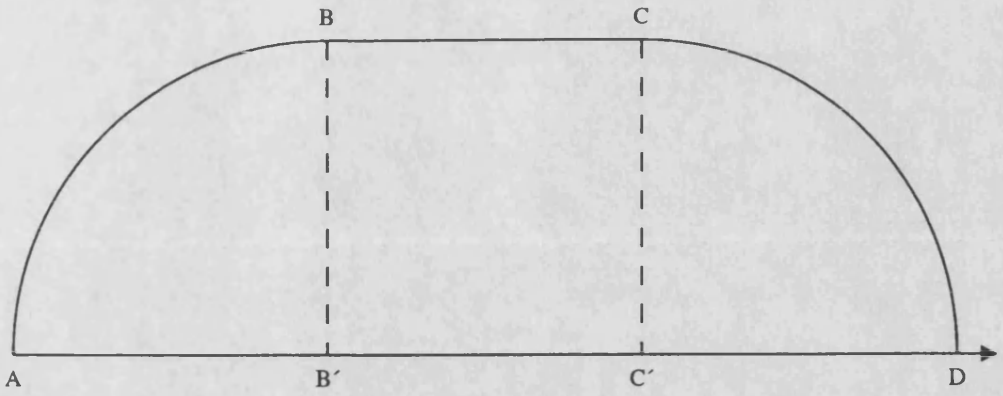


Figure 3.8. Motion Graph for Sinusoidal Fairing

$$AB' = \frac{2AB}{\pi}$$

Consider a point moving at a constant speed over ABCD. If this point is projected onto AB´C´D, then the resulting motion will appear to consist of acceleration, movement at a constant speed, and then deceleration to rest. In other words the motion will be faired.

To implement sinusoidal fairing suppose that the point traverses AB in n frames, and BC in m frames. The steady speed reached after the in-fairing stage is greater than that reached if fairing is not used. This speed will be the distance covered along AB´C´D divided by some virtual number of frames, v. The value of v is calculated by

$$v = \frac{2n}{\pi} + m,$$

$$v = m + n - n + \frac{2n}{\pi},$$

$$v = (m+n) - \left[1 - \frac{2}{\pi}\right] n,$$

$$v = total\ frames - \left[1-\frac{2}{\pi}\right] fairing\ frames. \tag{3.4f}$$

If the same number of fairing frames, n, are also used at the out-fairing stage, then we obtain

$$v = (m+n)-2n\left[1-\frac{2}{\pi}\right]. \tag{3.4g}$$

We can thus evaluate the steady speed reached between B′C′. The frame locations during the in-fairing phase are calculated by applying equation (3.4a). Similarly the frame positions for any out-fairing phase are calculated by applying (3.4b).

3.4.3 Laws of Motion for Constant Acceleration

Another way of modelling the motion effects we require is to use the laws of motion for constant acceleration. Particularly appropriate to our needs is

$$s = ut + \frac{1}{2}at^2 \tag{3.4h}$$

where s is the displacement, u is the initial speed, a is the acceleration, and t is the time. By substituting the animator's defining conditions into this equation we obtain the value of the acceleration required over the motion segment. The individual frame positions of the moving object can then be calculated. Figure 3.9 depicts the distance against time graph obtained by using this technique on the same example motion definition as that used in section 3.4.1.

We automatically get a smooth interchange between motion segments as this equation allows for the initial speed of the object. A new constraint, however, is that since the object only travels around its path in one direction, the value of the object's velocity must not change its sign during the motion definition. When decelerating an object we found that in too many cases the animator's distance and time constraints could only be satisfied if the velocity did change sign. This is the case in the fourth motion segment of our example, the distance-time graph

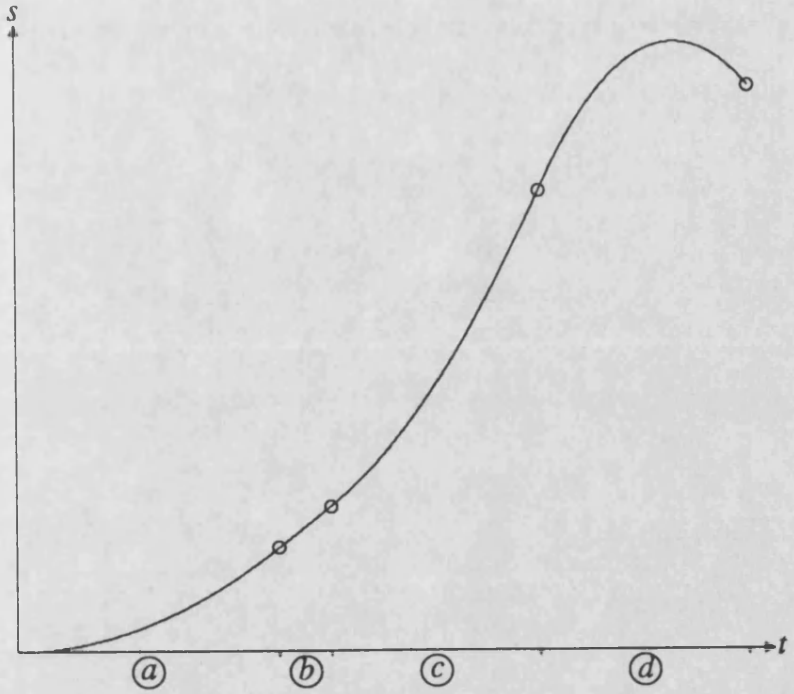


Figure 3.9. Using a Law of Constant Acceleration

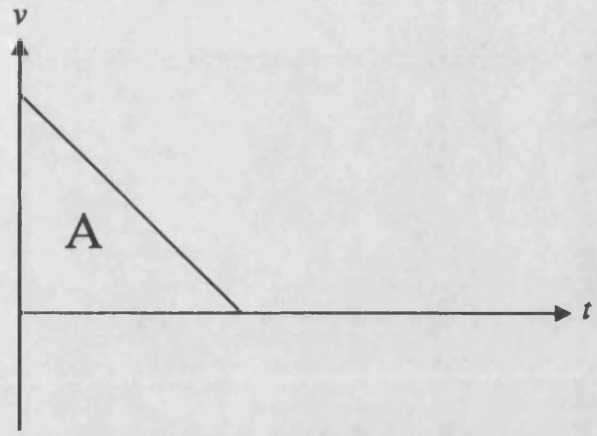
goes through a maximum. The reason for this is illustrated graphically in fig. 3.10. The motion segment in these cases has to be rejected.

3.4.4 Using an Arithmetic Progression

It is also possible to model acceleration and deceleration by using the formula for an arithmetic progression

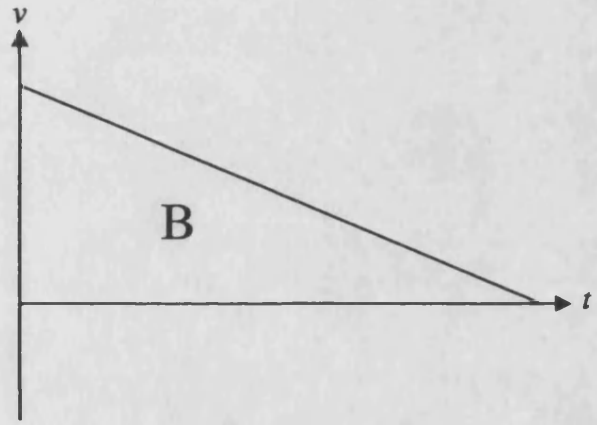
$$S = a + (a+d) + (a+2d) + \cdots + (a+[n-1]d) = \frac{n}{2}(2a + [n-1]d) \ . \quad (3.4i)$$

Making a the current speed ensures a smooth interchange between motion segments. This means that the penultimate position of the previous motion segment is used as the initial position of the new motion segment. An extra 'virtual frame' is thus incorporated and so the number of frames allocated for this motion segment should be incremented by one, call this n. By substituting these constraints into (3.4i) we obtain a value for d. The resulting motion effect is one of acceleration if d is positive, constant speed if d equals zero, and deceleration if



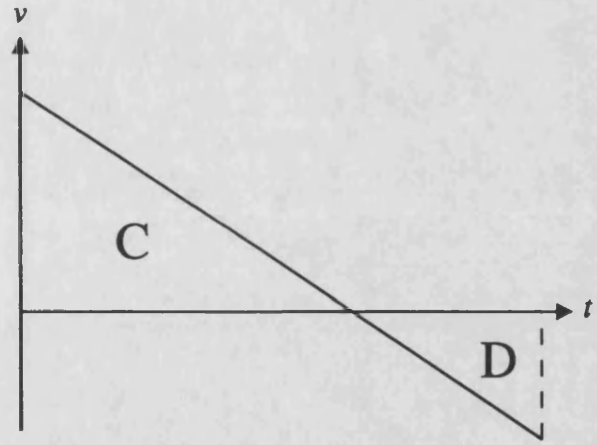
Graph 1: displacement, s = A;

Increase time taken
for the deceleration:



Graph 2: s = B > A,
but animator wants s = A;

To satisfy this:



Graph 3: s = C - D = A,
but v becomes negative!

Figure 3.10. Why Velocity can Become Negative

d is negative. The value of d is equivalent to the acceleration value in (3.4h) and so using an arithmetic progression is not really any different from using this acceleration law. The results obtained in section 3.4.3 also apply here.

3.4.5 Comparisons

To compare the methods used to achieve smooth motion we have superimposed the two distance against time graphs obtained in section 3.4.1 and section 3.4.3 (fig. 3.11).

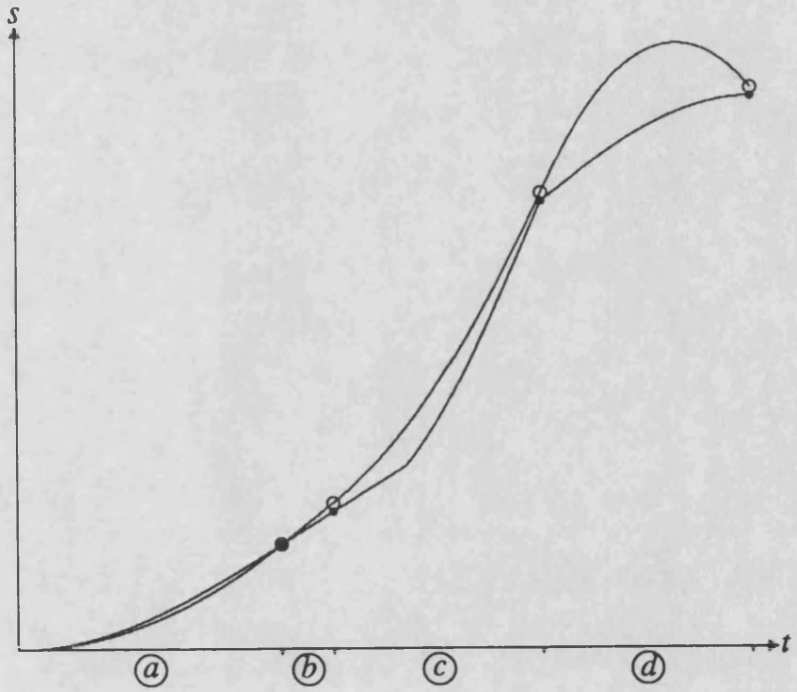


Figure 3.11. Comparing the Trigonometric Functions with the Acceleration Law

We can see that both methods give acceptable results when accelerating an object from rest. The final speed attained by the object differs, but this does not matter. We are not concerned with the actual value of the object's speed, only that the final motion is visually acceptable to the animator. Maintaining an object at a constant speed is straightforward in both cases. The law of constant acceleration, however, is far better at accelerating an object when it is already in motion. The initial speed of the object is automatically taken into account and so we do not

need to apply extra techniques to get a smooth interchange. As we have seen, there can be a problem when using the acceleration law to decelerate an object. In some circumstances the object can only satisfy the animator's constraints if it overshoots its destination and then comes back. We do not get this problem using the sine function, however. Fitting the motion to a sine curve ensures that the distance and time constraints are satisfied without a change in the direction of the motion. The cost is that the initial speed of the object is ignored and so we lose the smoothness of the interchange.

We decided not to offer the technique of sinusoidal fairing in Controller. This effect can already be achieved by combining motion segments obtained from the other techniques that we have described. We also have the problem of accelerating the object when it is already in motion. The formula for sinusoidal fairing does not take this into account.

Our best approach for achieving smooth motion has been to use a combination of (3.4a), (3.4b) and (3.4h). Priority is given to the law of constant acceleration, but where this fails the trigonometric functions are utilised. More complicated strategies would produce more realistic motion. However, by using the techniques described above we get an acceptable method of motion planning that is straightforward to carry out.

3.5 Techniques For Faking Mass

By carefully timing the motion of an object an animator can emphasise its size or weight (White 1986). He has to make objects move more slowly as they get heavier, and perhaps give them more difficulty in controlling their weight.

Using the previous methods of motion planning the animator can easily make two objects move at different rates. If both objects are to move from rest, the first motion segment of each will be an acceleration phase. At any given time during this period the total distance covered by the heavier object must be less than that covered by the lighter object. So if he wants both acceleration phases to last for the same amount of time he must ensure that the heavier object will

traverse less of its path. This distance is a constraint that the animator has to define when specifying a motion segment and so there is no difficulty in doing this. If the objects now proceed at a constant speed the lighter object will be travelling at a faster rate. The animator's judgement in indicating the distance of the acceleration phase will determine how convincing the final result will be.

One aim of our approach to computer animation has been to give the animator fine control such as he has in the above example. We do not want to make the specification too difficult for the animator and it will not be if we are only concerned with a few objects. However, if the scene is to contain many objects each with a different weight, such motion control could become a headache to the animator. He has to keep track of how heavy all the previous objects were and fit in new objects accordingly. It might be easier if the animator just estimates the weight (or mass) of each object and lets the animation system take care of the rest. We therefore provide a facility to do this and again avoid the use of computationally expensive dynamic methods.

When the animator selects a cast member he has the option of defining its mass. The unit of mass is immaterial as we only need to depict the relative mass of the objects in the scene. We now have to satisfy three user-defined constraints (mass, distance and time) and so the complexity of the problem increases. The purpose of this exercise, however, is to save the animator from having to remember all the distances he has been using to emphasise the mass of each object. So we let Controller work out how much of the path will be traversed and just get the animator to specify time and mass.

3.5.1 Utilising Existing Methods

When we use equation (3.4c) to model distance travelled under acceleration, the result is scaled by the total length of the motion segment. This segment length must now be determined by Controller using a function that depends on the mass of the object and the duration of the motion segment. The same is true if we are using the law of constant acceleration (3.4h), the segment length must now be calculated by the system and not defined by the animator. The segment length

should increase with time but decrease as the mass of the object increases, so

$$segment\ length\ \alpha\ \frac{time}{mass}\ . \tag{3.5a}$$

The simplest relationship satisfying this condition is

$$segment\ length\ = k \times \frac{time}{mass}\ ,\quad k\ constant. \tag{3.5b}$$

We let the animator determine the value of k by getting him to define the distance he would expect an accelerating object of unit mass to cover in some specified time interval. This task is performed by the animator before the path planning stage of our system. He uses a graphical valuator to input the required distance. The animator is thus still in general control of the final motion effects achieved.

The distance-time graphs obtained by doubling the mass of an accelerating object whilst keeping the other constraints fixed are given in fig. 3.12.

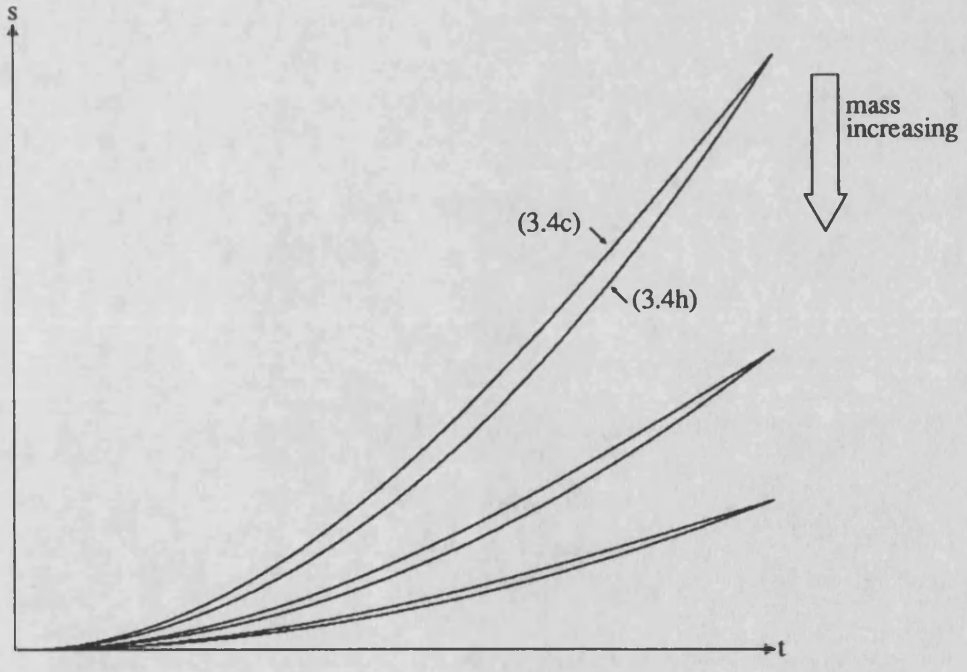


Figure 3.12. Faking Mass using Existing Techniques

Here the animator has already indicated the distance that an object of unit mass is required to cover and Controller automatically calculates the other distances. In

both cases the total distance covered in a fixed interval of time is halved as the mass of an object doubles. This is what we would expect if the objects are accelerating from rest.

The technique of sinusoidal fairing can also be adapted to take into account the mass of an object. For an object that accelerates to some steady speed, the number of in-fairing frames allocated should be proportional to the mass of the object. As the mass increases so should the number of fairing frames used. The animator specifies the total number of frames required for the whole of the motion segment. Sometimes, however, the mass of an object is such that more in-fairing frames are required than the total number of frames allocated by the animator. A maximum must therefore be placed on the proportion of the total number of frames that can be used for fairing. Unfortunately this means that the motion effect achieved for objects over a certain mass is the same.

3.5.2 Using a Family of Acceleration Curves

We decided to look for other ways of modelling the motion of objects with different mass to see if we could improve on the above methods. For example, the function

$$y = \frac{x^2}{ax^2 + b}\ ,\ a\ and\ b\ are\ constants, \tag{3.5c}$$

has been experimented with. By varying the values of a and b a family of curves can be produced (fig. 13a. and fig. 13b). We tried

(i) varying the value of a whilst keeping b fixed;

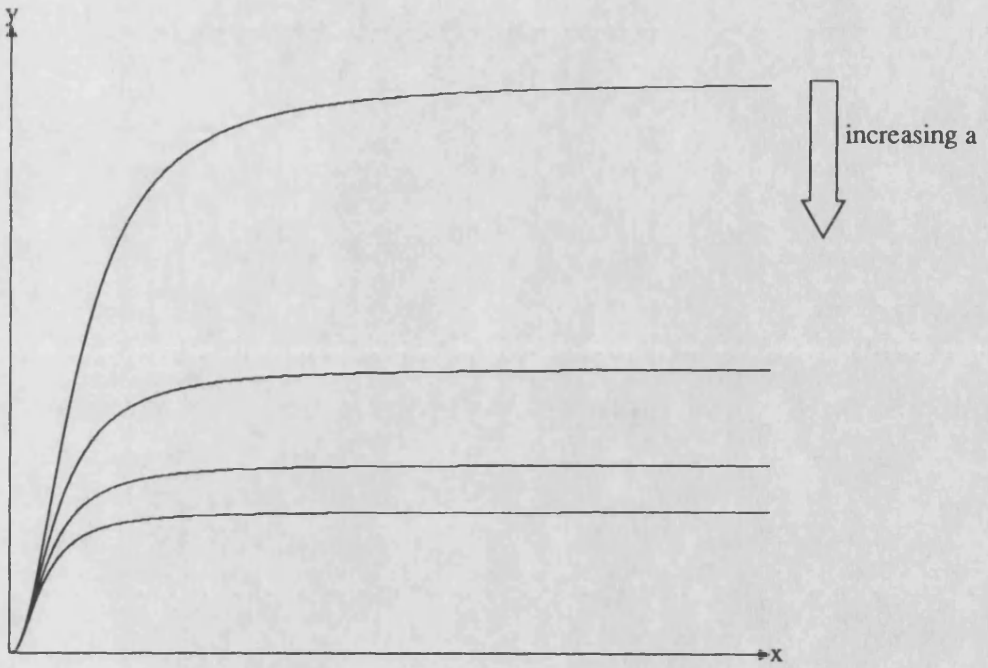
(ii) varying the value of b whilst keeping a fixed.

We then considered what the motion effect would be if these graphs represented plots of velocity against time, that is

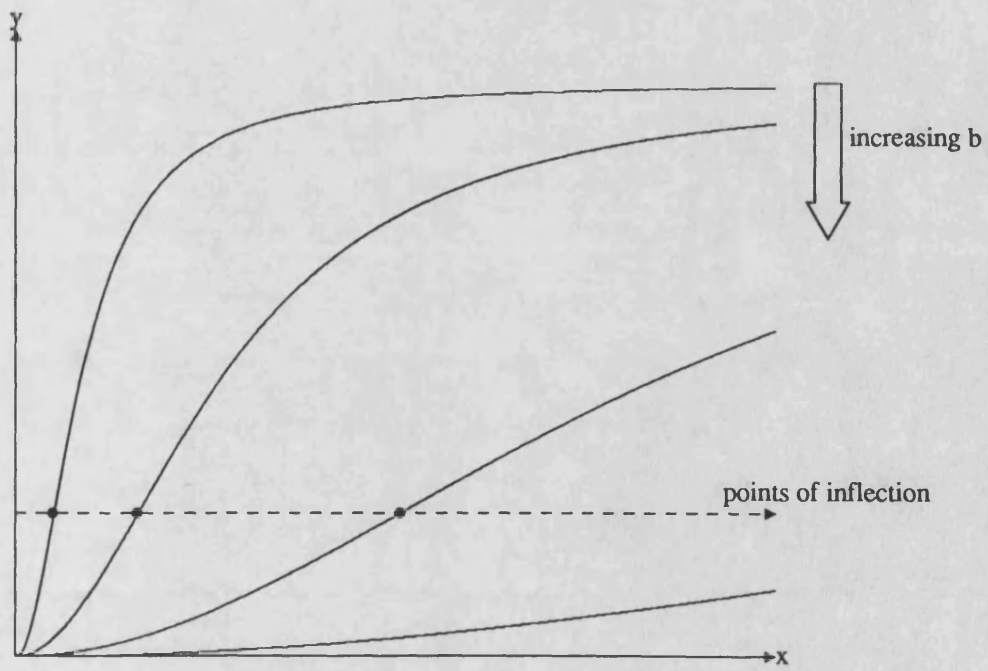
$$v = \frac{t^2}{at^2 + b}\ ,\ v = velocity, t = time. \tag{3.5d}$$

Each plot has the same general shape. Up to the point of inflection* the moving

* This occurs at $\left[\sqrt{\frac{b}{3a}}, \frac{1}{4a}\right]$.



a. Varying the value of a whilst keeping b fixed



b. Varying the value of b whilst keeping a fixed

Figure 3.13. The Family of Curves produced from (3.5c)

object can be thought of as overcoming its inertia. It then proceeds to accelerate, tending toward some maximum velocity. Increasing the value of a decreases the maximum velocity obtainable by the object. This suggests that a is related to the force that propels the object. So if a is fixed each object is propelled by the same amount of force. Increasing the value of b then increases the time taken for the object to reach any given velocity; up to the maximum velocity obtainable. So b appears to be related to the weight or mass of the object.

We can get a distance against time function by integrating over time.

$$s = \int \frac{t^2}{at^2 + b}\, dt \ , \ s = displacement, t = time.$$

$$s = \int \frac{1}{a}\, dt - \int \frac{b/a^2}{t^2 + b/a}\, dt \ ,$$

$$s = \frac{t}{a} - \frac{b}{a^2}\left[\sqrt{\frac{a}{b}}\tan^{-1}\left[\sqrt{\frac{a}{b}} \times t\ \right]\right],$$

$$s = \frac{t}{a} - \frac{\sqrt{b}}{a\sqrt{a}}\tan^{-1}\left[\sqrt{\frac{a}{b}} \times t\ \right]. \qquad (3.5e)$$

We need to determine suitable values for a and b. In our current implementation we emphasise the mass of the object and so keep a fixed at $a=1$. Equation (3.5e) can then be simplified to

$$s = t - \sqrt{b}\tan^{-1}\left[\frac{t}{\sqrt{b}}\right]. \qquad (3.5f)$$

Although b is proportional to the mass of the object we do not set b to be equal to this mass. In fig. 3.14 the value of b has been successively increased by a factor of 10. The mass defined by the animator is typically in the range one to one hundred. However, if the value of b is also in this range then the object rapidly reaches its maximum velocity, too rapidly to be usable in Controller. To obtain the best results we found that the value of b should be much larger. Multiplying the mass by a factor of 10^5, for example, produces a more gradual

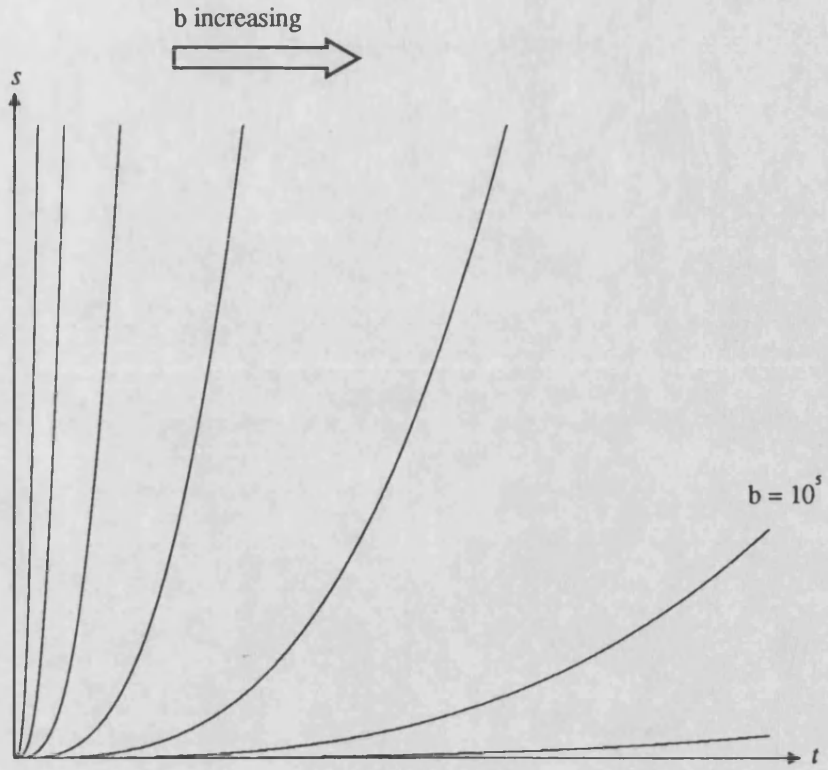


Figure 3.14. Using a Family of Acceleration Curves

increase in velocity and is more suitable for our purposes.

The period of time between successive frame positions is constant and is used as the unit of time in Controller. Time can therefore be represented as integer values starting from $t=1$. However, using a value of b in the magnitude of 10^5 means that s is too small until many time units have elapsed*. Again this is not suitable for Controller's purposes and so s must also be scaled. Maintaining our approach of giving the animator as much control as possible, it is he who determines the value of the distance scale factor. He specifies the total distance that an object of unit mass will cover over a specified interval of time.

We also have to allow for the deceleration of an object. To do this we keep the time argument used by the distance function separate from the total time elapsed. The latter is always incremented as new frames are defined by the

* For example, when $t=1$ and $b=10^5$, (3.4f) returns $s=0.000003$.

animator. We only increment our distance function time, however, when the object is accelerating. When the object is decelerating it is decremented, and when the object is moving at a constant speed it is left unaltered. We thus travel back down the distance time curve when decelerating and a correspondingly smaller inter frame distance is calculated. This method will give a smooth interchange between successive motion segments. A drawback, however, is that the object's rate of deceleration will be the same as its rate of acceleration. The animator cannot, therefore, make an object decelerate over a period longer in time than it accelerated in. We can overcome this by varying the time intervals used with the distance function, but will then lose out on the smoothness of the interchange between motion segments.

3.5.3 Comparisons

The motion graphs given in fig. 3.15 provide a comparison of the two methods that we have described for faking mass.

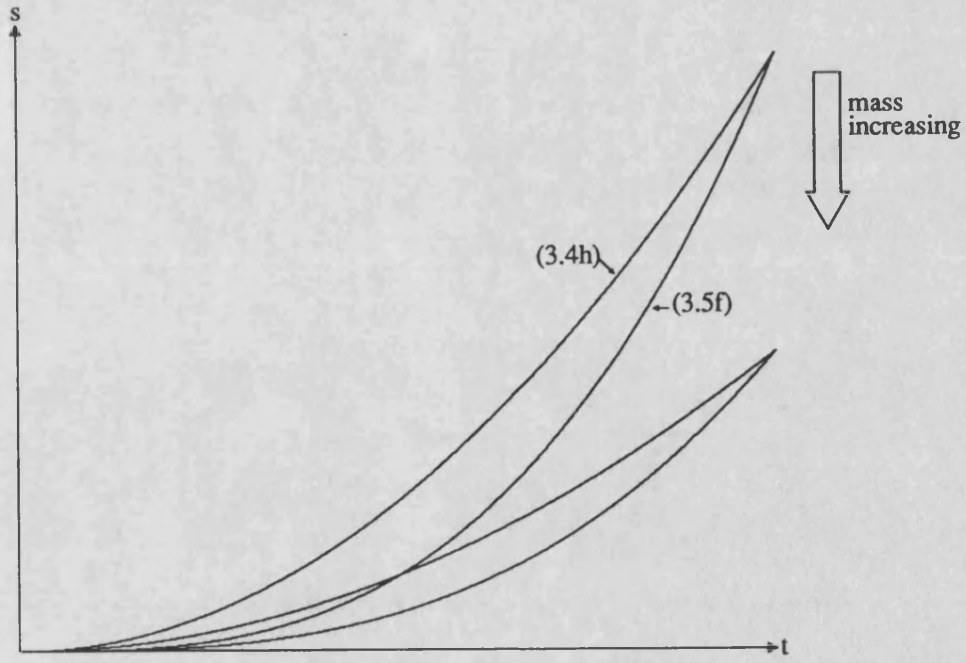


Figure 3.15. Comparison of the Methods used to Fake Mass

To aid in this comparison we have made sure that the final distance covered is the

same in both cases. We can see that when (3.5f) is used, an object accelerates more slowly to begin with, and consequently gives us a better feel for the inertia of the object. It thus reaches a greater final speed in covering the same distance in the same time than when the acceleration law is used. Note that in both cases the total distance covered appears to be halved as the mass of an object doubles. This is not true when using (3.5f), but when large values are used for b (as in our implementation) the differences are not significant.

We have preferred to incorporate mass by using the family of acceleration curves. A motion graph resulting from this method is an intuitively better approximation to the real thing. We also have a way of manipulating the force applied to an object built into this distance function. This facility has yet to be exploited.

3.6 Verifying the Motion Definition

In general the motion segments defined by the animator can be combined in any manner. He is thus in control of a flexible system of motion planning and can move an object in many different ways. Such flexibility, however, may produce anomalies in the motion definition. Controller therefore provides a *motion verifier* to determine how well each motion segment fits in with the current state of the overall motion. If a good fit can be made then no action is taken. Otherwise, warning or error messages are issued, and these are detailed below.

3.6.1 Warnings

The animator is given a warning if the resulting motion is unrealistic or cannot be fitted in smoothly. He can either ignore this warning (as he may really want such an effect), or repeat the specification for the motion segment concerned. Warnings occur, for example, when:

(i) A stationary object is made to move at a constant speed without first accelerating it from rest. The facility to do this is needed, however. A cast member, for example, may enter the scene already travelling at a constant speed.

(ii) The object is to continue its motion at a constant speed. However, the distance that can be covered in one unit of time by maintaining this speed is greater than the total length of the motion segment defined. Here one frame position is plotted and the object's speed is decreased accordingly.

(iii) Deceleration is required but the constraints cannot be fitted in smoothly (see §3.4).

(iv) The object is held at its present position for several frames but its current speed is greater than zero. Here the object is coming to a sudden halt. Note that this is acceptable if the current speed of the object is small enough so that the object may be stopping anyway. The warning is only given if the speed is greater than some threshold value above which it is unrealistic for an object to stop in a single frame.

3.6.2 Errors

Some motion segment specifications will be impossible to carry out, but these are usually obvious:

(i) A stationary object is asked to decelerate.

(ii) Deceleration in one frame is required but the segment length is greater than the distance that can be covered by the object if it maintains its current speed. Here the object can only satisfy the distance and time constraints by accelerating.

(iii) An object is asked to accelerate over a distance that is less than the distance it can cover in one time unit by maintaining its current speed.

3.6.3 Using the Automatic Mass Facility

When the automatic mass facility is being used much of the above motion verification is not applicable. The animator does not specify the distance of the motion segment here. Note that the mass of an object is emphasised when objects are accelerating from rest. If an object appears on the scene already travelling at a constant speed then using the automatic mass facility is not as effective. In these circumstances the animator is warned of this fact.

3.7 Summary

In this chapter we have described a procedure that allows an animator to specify the overall motion of an object about a set. The motion path generated for this purpose is made up of a spatial definition (in two dimensions), and a temporal definition. The former involves drawing the track that the object moves along whilst the latter involves the specification of a series of motion segments. Each motion segment describes how the object will move over a particular section of its track.

With both definitions we have attempted to provide the animator with a flexible interface offering him fine control over the motion specification. The interface should not be impossible to use, however. The animator should find that moving an object along its path is as natural as driving a car along a road. For the latter decisions about whether to slow down, speed up or stop depend on the road conditions and the driver's destination. In the animation case the decisions are similarly governed by the storyboard that describes the scene. We have found that the ease of use of the interface decreases as the complexity of the problem increases. We therefore endeavour to keep the implementation as simple as possible. For example, we have used kinematics rather than dynamics to implement the temporal definition.

The motion specification is not yet finished. Several parameters are associated with an object at each frame position that has been defined. We will now go on to look at how the animator can specify the value of these parameters.

Chapter 4

Parametric Animation

4.1 Introduction

In chapter 2 we describe how to represent objects using an appropriate set of parameters such as

Cast member = {*location*, *orientation*};

Camera = {*location*, *orientation*, *zoom*};

Light = {*location*, *intensity*}.

The motion path defined by the animator provides the location (in the ground plane) of an object at each frame of the animation. He must now complete the specification of the above parameters and so provide the configuration of the object at these locations. In this chapter we will detail Controller's version of *parametric key frame animation* that allows the animator to do this.

We will also describe the remaining facilities that Controller offers to the animator. They include a computer version of the pencil test used in conventional animation, and methods for generating the frames.

4.2 Specifying the Parameter Values

Having defined a motion path, the animator begins specifying the parameter values associated with the camera, cast member, or light to which it belongs. He does not have to specify these values at every frame location, however, as this would be far too tedious. The advantage of parametric animation is that parameter values can be interpolated across a range of frames. Note that the

animator can also go back to a previously defined object and refine its parameter values.

The animator follows the same procedure regardless of the parameter being specified. First he selects (on screen) the two frame locations between which a parameter value is to be computed. The rectangles in fig. 4.1 are centred on the frame locations that have been chosen in this way.

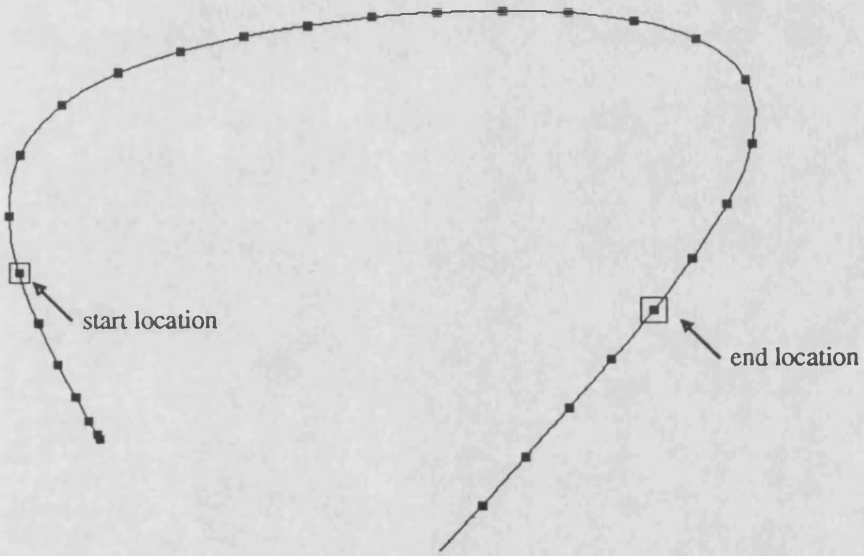


Figure 4.1. Selecting the Frame Locations

Each location is defined to have a small *gravity field* surrounding it so that the puck does not have to coincide exactly with the screen coordinate of the selected frame. The puck position may fall within the gravity field of several frame locations, however. If this occurs then the distance of each frame location from the puck is calculated so that the nearest one can be identified. If the required frame still cannot be determined then the animator is informed of the possible candidates and requested to use the keyboard to select the one he wants. A keyboard entry is always needed when an object is at the chosen location for more than one frame.

Chapter 4. Why leave height until this stage?

Why ~~can the~~ lens use the range [10°, 170°] for lens angles?

N.B the def^n configuration of the viewer as used by the ray tracer requires a 56° lens angle - already more than physical cameras!

The value required for the parameter at the second frame location is now input using an appropriate graphical valuator (these are described below). The parameter value at the first frame location is not specified at this stage, however. Controller uses its existing value or, if this has not been defined, the value found at the preceding location that is nearest to it in time. This helps to ensure that there are no sudden changes in the motion affected by this parameter. The techniques used to calculate the frame locations along a motion path (chapter 3) are now used to interpolate between these two values. The animator chooses the method that will give him the effect that he is after. The use of splines and physical laws are again avoided when carrying out the interpolation. We endeavour to keep parametric animation as straightforward as possible and so do not use the more complicated strategies.

The animator can also specify the value of a parameter at a single frame location only. He has to carry out the above process but selects the same frame location as the start and end points. The value then input using a graphical valuator becomes the value of the parameter at this frame. Let us now examine in more detail the parameters that express the cameras, cast, and lights.

4.2.1 Height

To complete the definition of a three dimensional path the height of the object must be determined. The object is assumed to be at the level of the ground plane until the animator carries out this task.

Following the above process, the animator uses a 'sliding scale' valuator (fig. 4.2 and colour plate 3) to define the object's height at the second frame location selected. The criterion for calculating the height at the inbetween frames is then chosen from the following:

Linear The total height difference is divided equally between all the frames involved.

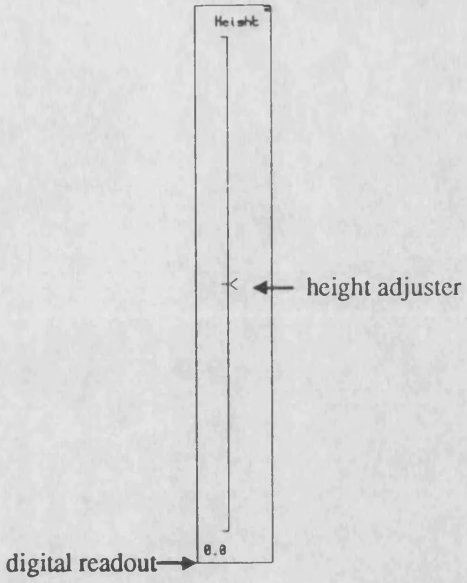


Figure 4.2. The Height Valuator

Faired The height difference is divided so that successive increments are increasing at the beginning, and decreasing at the end. Equation (3.4a) is suitable for this task.

Gravity Successive increments in the height are calculated as if the object is falling or rising under gravity. More details of this option are given in chapter 5.

Constant All the inbetween frames are assigned the same height value.

The height of an object is more conveniently defined in this way than at the track drawing stage. The latter would require the animator to draw a three dimensional path using a two dimensional device. Providing him with an easy to use method for achieving this is not straightforward. Here, however, the animators task is less complicated and effective results are produced.

4.2.2 Orientation

The animator has to determine the orientation of an object along its motion path. For example, the camera head* must be adjustable so that it can point in any desired direction in three dimensional space. We will describe separately how the cameras, cast members and lights can be orientated by an animator.

(i) Cameras

A camera moves in the left-handed coordinate system used to generate the set models (see §2.3.1). Initially the camera head is assumed to point along a straight line parallel to the z axis and in the direction of positive z. The animator can, however, select two frame locations and alter the orientation of the camera head between them. The following options, for example, enable him to rotate the camera in the ground plane away from its default heading:

Point The animator selects on screen a position in the set at which he requires the camera to point. The camera is then directed towards this point from each of the specified frame locations.

Fixed The 'dial' valuator (fig. 4.3) fixes the camera head at a constant angle to its default heading. The dial will show the angle of the camera head from its default position at the start frame location when it is initially displayed on the graphics screen.

Track The animator uses the puck to select a motion path belonging to a cast member. The camera is then centred on the location of this cast member at each frame.

Revolve The camera is made to turn through some angle determined by the animator. He also specifies whether the rotation is to be clockwise or anticlockwise.

---

* The camera head is the term used when the view position and the view direction of the camera are considered as one entity.

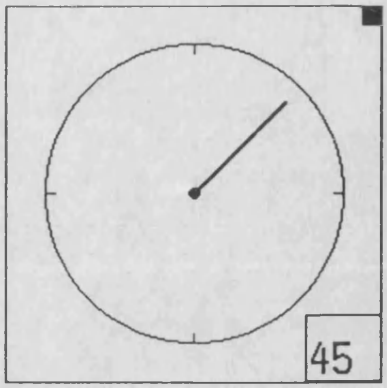


Figure 4.3. The Dial Valuator

Tangent The camera is made to follow the tangent to its motion path at each frame location, or to be at some fixed angle to this tangent. The dial valuator defines the value of this angle.

Pan The camera is interpolated from it initial heading to some new heading defined by the animator.

All these methods deal with just one parameter; the angle of the camera head from its default position. For the **revolve** and **pan** options, this angle is interpolated across the inbetween frames using the technique considered the most appropriate by the animator. A short line is also plotted from each frame location in the direction in which the camera head now points. This is demonstrated in fig. 4.4 where the animator has just used the **point** option. He thus has a visual impression of the camera's heading.

As well as rotating the camera head in the ground plane, the animator can make it tilt. A default tilt angle of zero is assumed and this corresponds to the camera head being parallel to the ground plane. The valuator illustrated in fig. 4.5 then enables the animator to define the tilt angle he requires. The two end values are interpolated across the inbetween frames.

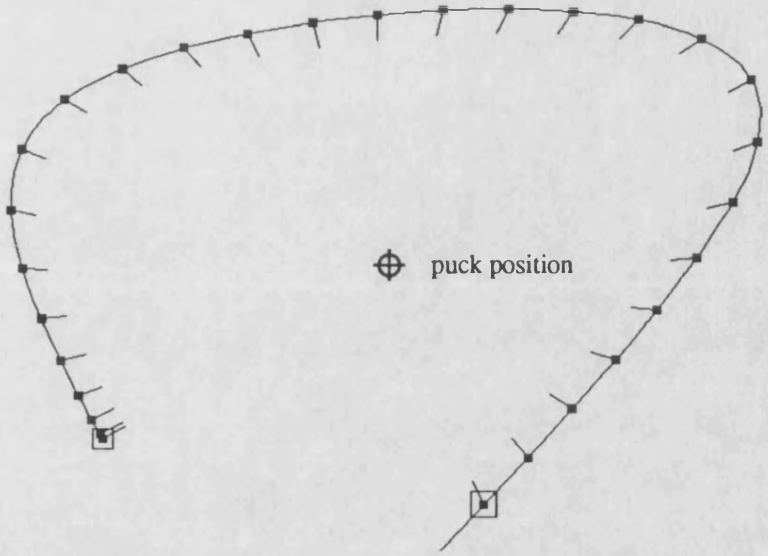


Figure 4.4. 'Pointing' the Camera Head

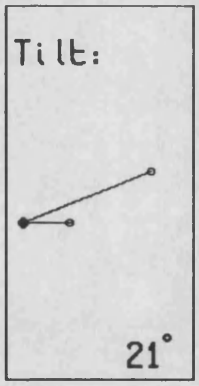


Figure 4.5. The Tilt Valuator

(ii) Cast

The internal motion of each cast member is portrayed by using an appropriate set of poses (see §2.3.3). Hence, when using Controller to plan the motion of the cast, they can be considered as rigid bodies. The animator's task is thus greatly

simplified and most of the functions used to manipulate the orientation of a camera are reusable here (the **track** facility is the obvious exception). A cast member can be made to follow the tangent to its motion path, revolve about some point, and so on.

When rotating a cast member, its current frame location is used as the origin of its coordinate system and each axis of rotation passes through this point. The centre of rotation of the object must also be specified, for example, it may be at the base of the object, or at its centre of gravity. This point is defined at the modeling stage and its placement depends on the cast member being designed.

(iii) Lights

A point light source radiates light in all directions and so its orientation is immaterial. A spot light, however, is restricted to radiate light in the particular direction defined by the animator. The functions for determining the camera's heading are again suitable for this purpose. The **track** function, for example, will define a light to follow a cast member around a set just as a real spot light follows an actor around a stage.

4.2.3 Camera Zooms

A camera can obtain close up shots of a scene by moving nearer to the objects in that scene, and distant shots by moving away from these objects. Often, however, physical restrictions are imposed on the movement of a television camera and the desired shot can only be obtained by using different lens angles. Typically, a lens angle of between 10° (for close ups) and 50° (for distant shots) will be used by the cameraman (Millerson 1973). A television camera is equipped with a *zoom lens* that allows the required angle of view to be varied with a smooth continuous action. Narrowing the lens angle in this way is termed *zooming in* whilst widening the lens angle is termed *zooming out.*

Although the virtual cameras used in Controller do not suffer from as many restrictions as television cameras, a zoom facility is still useful. A virtual camera,

for example, must be located somewhere on the set and so it may be impossible to obtain the required shot without using a zoom. As before, the animator selects the frame locations between which the lens (or view) angle of the camera is to be changed. The valuator illustrated in fig. 4.6 then allows him to define a view angle at the end of the zoom of between 10° and 170°. The range available for this view angle is much larger than that obtainable with a television camera. Controller also supplies the animator with a wire frame view of the shot obtained from the camera. In fig. 4.6, for example, the animator is adjusting the view angle so that a zoom out is obtained and the area of the scene covered by the camera shot is increased. Just as when using a real zoom lens, the zoom process should occur smoothly. Equation (3.4a) is again suitable for the interpolation of the view angle parameter across the inbetween frames, although other methods can be used if desired.

4.2.4 Rotating the Camera Shot

By using image inverter prisms the shot obtained from a television camera can be made to rotate (Millerson 1973). This special effect is also available in Controller. The animator uses the dial valuator to define the angle of rotation to be applied to the camera shot. This angle is then interpolated between the specified frame locations, usually with a faired motion.

The required effect is achieved during the rendering process. The image rotation angle is applied to the screen on which the rendered image is formed.

4.2.5 Light Intensity

Controller allows the animator to position light sources about the set currently being used*. The intensity of each of these lights must be defined by the animator and the valuator depicted in fig. 4.7 is used for this purpose. The intensity is defined to fall between zero (dim) and one (bright). The animator can

* Usually these lights will be stationary throughout the scene, but can be defined to follow a motion path if desired.

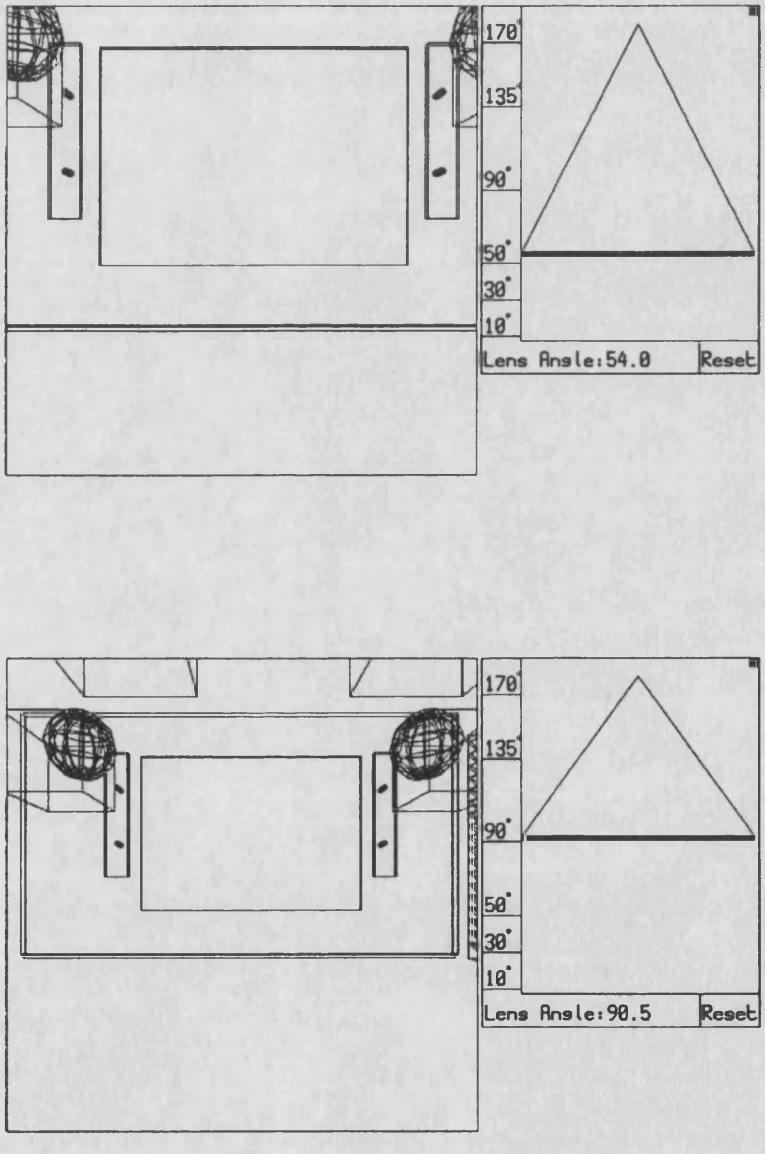


Figure 4.6. Changing the Angle of View

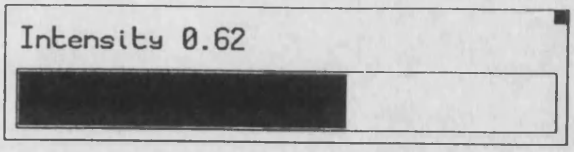


Figure 4.7. The Light Intensity Valuator

also request a light to be faded in to a certain intensity and faded out from a certain intensity. For the former the intensity is increased from zero over the specified frames, whilst for the latter it is decreased to zero.

4.3 Verifying the Animation

The animator will need to preview the results of his animation planning and make any adjustments that he deems necessary. In this section we will describe how Controller aids the animator in this task. Note that the facilities provided can be used with any of the motion paths currently plotted onto the set.

4.3.1 The Status Function

A digital read out of an object's status at a frame location selected by the animator is useful. He can then determine whether a particular parameter has been defined. He may also want to obtain the value of a parameter at this location so that he can use the same value at other locations. This will help to keep the motion flowing smoothly.

When using this **status** option, the required information is displayed in a pop-up window. The window in fig. 4.8, for example, lists the parameter values found at a frame location belonging to a camera. A wire frame view of the camera shot at this location is also displayed on the graphics screen.

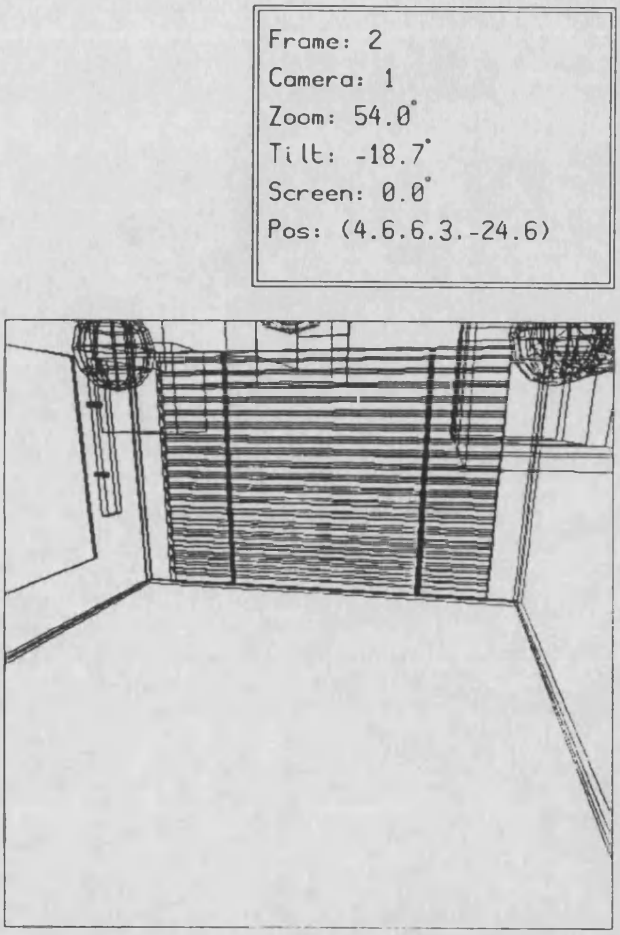


Figure 4.8. The Status Function in Use

4.3.2 The Pencil Test

When developing an animation it is important that the animator can preview each action in real time. A facility similar to the pencil test used in conventional animation is provided for this purpose. The animator selects the range of frames over which a computer animation pencil test is to be performed. An animated line drawing is then calculated frame by frame and presented to him on the graphics screen. The pencil test can be of either a cast member or of a camera. If the object is a camera then the scene at each frame location along the path is displayed so that the animator can verify that the camera shot is as he desires. At this stage none of the cast members are shown. If the object is a cast member it is also necessary to specify the camera it is to be viewed from. The sequence then presented is that resulting from animating both this camera and the cast member for the selected frames.

Ideally, all cast members currently in the scene are displayed during the pencil test. However, even if only a few cast members are drawn then the generation of each frame takes several seconds. This makes the pencil test far too slow to be useful in an interactive animation system. We therefore restrict the pencil test to show the set and selected cast member only. The motion of an individual cast member can then be studied in detail. The interaction between several cast members can be viewed later using a real time playback facility (see §4.6) that displays pregenerated images at an appropriate rate. If the animator wants to do this at this stage then he must wait (perhaps for several minutes) for the selected frames to be generated.

To present a pencil test in real time, twenty four or twenty five frames have to be displayed every second. To calculate each line drawing as quickly as possible, hidden line removal is not considered. Also, wire frame representations of the sets and cast are generated before Controller is invoked and often use a simpler model of these objects. For example, a stick figure representation may be good enough to represent some cast members. Only the configuration defined by the animator then has to be calculated and applied. We are ultimately restricted by the hardware available, however, and Controller only manages to generate up

to three frames every second. The animation is thus presented in slow motion but despite this we still find the pencil test facility useful. Another possibility is to plot only the vertices of the line drawings as the resulting sequence may still give a feel for the motion. The time savings here, however, are unlikely to compensate for the loss of detail in the sequence, but may be worth investigating.

The main disadvantage of Controller's pencil test is that it fails to indicate the timing of a motion. One way of providing for this is to calculate the frames after longer intervals of time. For example, if Controller calculates only every eighth frame then, although the resulting sequence will appear to 'flicker' badly, it is displayed in the correct time. Providing the animator with the option to miss out frames in this way could be the most flexible method of providing a pencil test facility.

4.4 The Animation Data

Once the animator has completed a session with Controller, the animation data is recorded in an appropriate file such as that represented in fig. 4.9. It contains

(i) the name of the set model used;

(ii) the number and initial frame of the scene;

(iii) the parameter list of every camera defined;

(iv) the parameter list of every cast member defined;

(v) the parameter list of every light defined.

A data file will not usually be as short as the one in our example (fig. 4.9). Many frames are required for an animation and so the parameter lists of the objects involved will be much longer.

All data files created by Controller can be loaded back into the system so that the animator can refine the animation and include new objects. When satisfied with his specification, the animator uses the data to generate the frames of the animation.

```
/graphics/Controller/Sets/studio

Scene= 1
Initial frame= 1

Camera= 1 4 Camera data:

471 0 210 470 229 0 0.000000 0.000000
471 0 210 470 229 0 0.000000 0.000000
471 0 210 470 229 0 0.000000 0.000000
471 0 210 470 229 0 0.000000 0.000000

Object= 3 4 Object data:

462 -60 542 0.000000 0.000000
462 -55 542 0.026877 0.000000
462 -40 542 0.107047 0.000000
462 -16 542 0.239140 0.000000

Object= 6 1 Object data:

464 -60 541 0.000000 0.000000
```

Figure 4.9. An Animation Data File

4.5 Generating the Frames

The animation frames can be generated using either a mesh renderer or a ray trace renderer. The former method is fast and is useful for previewing the animation. Unlike the pencil test facility, all the cast members defined are now included in the animation. Ray traced images are computationally expensive to calculate and so are only produced when the animator is certain that the animation is correct. The object and lighting models required to define the scene at each frame are calculated and then rendered by the ray tracer as a batch job. Other rendering techniques may be required, however, and will need to be supplied with an appropriate interface to the animation data file. So far we have only provided this interface for the two in house rendering systems mentioned here.

Once the rendering system has been selected, the animation data is summarised on the visual display unit. For example, a summary of the data file

listed in fig. 4.9 is given below (fig. 4.10).

```
Set: /graphics/Controller/Sets/studio      Scene: 1

Cameras:

          Code    Start frame    End frame
            1           1             4

Cast:

          Code    Name          Frame Duration
            1     dice                4
            2     tumbler             1

Next Shot is for frame 1;
What camera is to be used? _
```

Figure 4.10. Summary of an Animation Data File

The animator now defines his final requirements for the rendering of this animation sequence. He begins by selecting the order and duration that each camera shot is to be used. The viewer and screen positions used by the rendering system are calculated from the data found in the parameter list of the active camera. The area of the image obtained from a camera shot is known as the field size (White 1986). In both film and television a height to width ratio of *36:50* (the *academy field ratio*) is normally used for this field size. The animation frames produced in our system can also be rendered using this ratio. The animator specifies the width of the frames and their height is calculated accordingly.

The animator next decides the frames at which the cast will enter and exit the scene. They can be 'cued' at any point in the action and 'cut' from the scene even if all the frames specified for them have not yet been exhausted. The scale and colour scheme to be used for each cast member are also defined at this stage. The validity of frame durations specified by the animator throughout the above process are checked, and he is informed whenever a discrepancy occurs.

The rendering system also requires the following information (determined at the modeling stage) about each member of the cast;

(i) the number of poses used to define its motion;

(ii) the number of primitives used in the composition of each pose;

(iii) the centre of rotation of the composite cast model (see §4.2.2).

The motion poses are used cyclically at successive frames. The rendering system uses the animator's specification to update the original configuration of the pose currently being used. Each primitive it contains is therefore transformed as follows:

(i) it is scaled to the size specified by the animator;

(ii) the heading and tilt rotations are applied (in its own coordinate space);

(iii) it is translated to the set location defined for the current frame.

If the mesh renderer is being used then a line drawing of each cast member in its resulting configuration is plotted onto the frame image (using a perspective projection). Alternatively, if the ray tracer is being used then the resulting description of each cast member is appended to the object model being created for this frame. The process is then repeated for all the cast members that are present in the frame. Lighting information is only required by the ray tracer and, if not explicitly defined by the animator, is obtained from the original model of the set being used.

4.6 Real Time Playback

After an animation sequence has been rendered it is possible to view the frames using real time playback (see §1.6.8). A facility called **video** is provided for this purpose. The rendered frames are stored in the main memory of the host computer from where they can be quickly copied to the graphics screen. By displaying successive frames on top of one another the illusion of motion is produced in the same way as it is in projected film.

The frames are cycled continuously (in reverse order if required) and the time taken for the completion of each cycle is given to the animator. He can thus

calculate the pause required between successive frames that will produce the appropriate animation rate. This time delay can then be incorporated into the display cycle. The size of the rendered frames must be small, however, if the system is to cope with a rate of twenty four or more frames every second. Each frame can also be displayed several times in succession and so holds and double framing (see chapter 5) are catered for.

When using **video**, the animator may notice some error or discrepancy in the animation. He will then need to study the suspect frames so that he can identify the exact point where this occurs. A facility called **storyboard** displays a specified range of frames on the graphics screen and can be used for this purpose. In fig. 4.11, for example, twenty frames are being displayed from an animation sequence that contains such a discrepancy. This enables the animator to discover that the 'clown' character is missing from the sixteenth frame of the sequence. He can now take corrective measures beginning with an examination of the data file for this animation.

4.7 Post Production

The rendered frames are transferred onto film or video and a soundtrack is applied. These stages to produce the final computer animation are hardware dependent and outside the scope of Controller.

Note that the pencil test and real time playback facilities supplied by Controller allow the animator to preview and refine the animation as desired. He can therefore be confident that the appearance of the resulting animation will be satisfactory before the film transfer stage takes place.

The soundtrack should be used throughout the development of the animation so that the action is correctly synchronised with the sound. We do not have the necessary equipment for producing a soundtrack, however, and so (as with many computer animation applications) sound is only provided during post production. The synchronisation of sound with computer animation is an area in need of research.

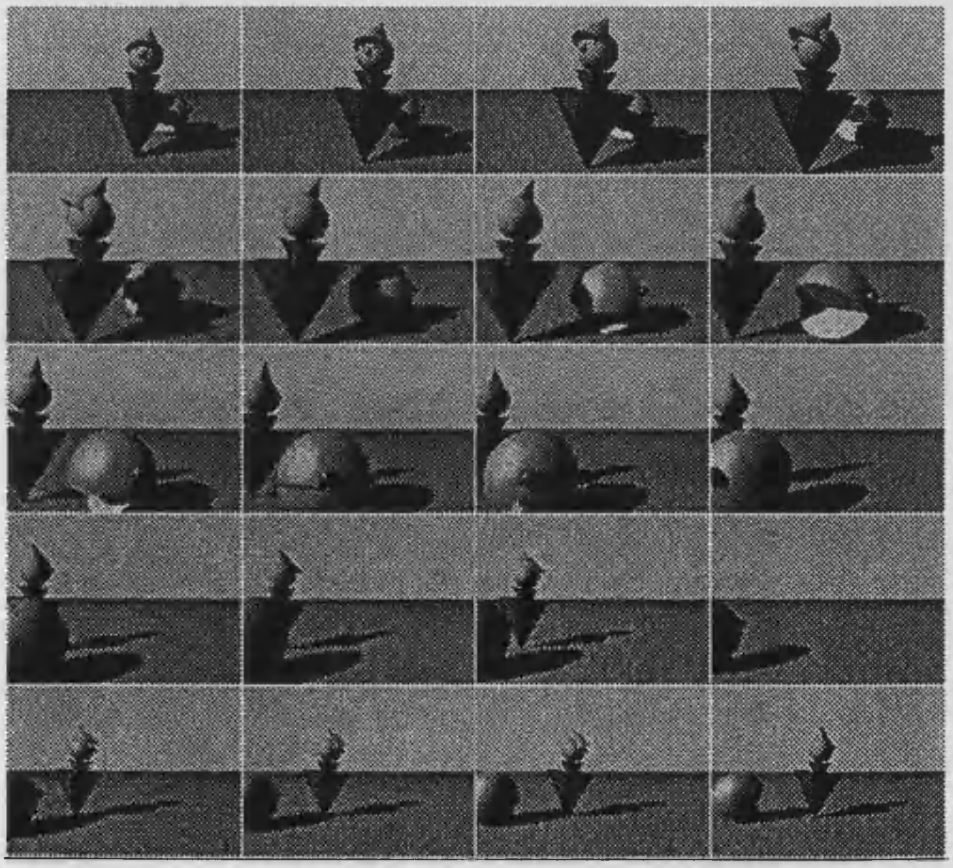


Figure 4.10. Identifying Rogue Frames

4.8 Summary

In this chapter we have described the process of parametric key frame animation as it is carried out in Controller. Our main objectives have again been to develop this process so that the animator finds it both flexible and straightforward to use. This is partly achieved by providing the same simple procedure for the specification of all parameter values. The animator is given a choice of the kinematic interpolation methods described in chapter 3. He can thus produce a variety of effects and ensure that the motion affected by a parameter occurs smoothly. The production of smooth flowing motion is further aided by interpolating from the existing value of this parameter.

Visual feedback is important in the development of any animation. Controller therefore provides a computer animation pencil test and facilities for previewing the animation using real time playback. The animator can thus refine the animation so that minimum editing will be required when it is finally transferred onto film or video.

Chapter 5

The Role of Traditional Animation Methods in Computer Animation

5.1 Introduction

In the previous three chapters we have described the development of the Controller animation system and detailed its use as a tool for the generation of computer animation sequences. Emphasis has been placed on making Controller both flexible and straightforward to use so that the animator is kept in control of the animation specification. This means that the skill of the animator remains an important factor. It can be argued, however, that it is difficult to define complex animation using Controller. We could overcome this by introducing greater automation into the system. Many systems for example use physical laws, such as the laws of dynamics, for this purpose (see §1.6.4). The amount of control that the animator then has will decrease, however, and so would be contrary to the aims of this thesis. We want to enhance the versatility of the animation that can be obtained without sacrificing the control of the animator. Our approach is to encourage the animator to utilise techniques taken from traditional animation and Controller will assist him in doing this wherever possible. This chapter will explain the advantages of such an approach and detail how we have carried it out.

A practical test is required to discover if Controller really is flexible and easy to operate. Controller was therefore made available to a graphic design student who used it to generate a complete computer animation. The results of this experiment and the experience obtained from it are detailed later in this chapter.

5.2 Why Use Traditional Animation Principles?

In chapter 1 we described the twelve principles most often adhered to by traditional animators. They are:

- staging;
- straight ahead action and pose to pose;
- slow in and slow out;
- anticipation;
- timing;
- arcs;
- follow through and overlapping action;
- secondary action;
- squash and stretch;
- exaggeration;
- solid drawing;
- appeal.

Most of these principles are useful in the generation of computer animation, particularly if the animation is for entertainment purposes. Their intelligent application can produce results that are just as convincing as those obtained from the use of physical laws. The computation cost will also be a lot less.

Controller already makes it possible for the animator to apply some of these principles. Using B splines to plot out the path of a moving object, for example, ensures that the object will move in an arc and not in a straight line (see §3.2.1). This helps to prevent motion from appearing too mechanical. The animator can also apply the principle of slow in and slow out when specifying the motion of some object. The techniques that we described in chapter 3 for supplying the temporal definition of an object can easily be used to 'fair' its motion in this way. We will now consider how we can develop this theme further.

5.3 Assisting the Animator

Several facilities have been introduced into Controller to assist in the specification of animation. Controller can, if required, take into account the mass of an object and calculate the effects of environmental factors such as wind and gravity. Objects and cameras can also be made to vibrate or 'stagger'. All these facilities have been designed to 'fake' reality rather then give an accurate simulation. Dynamic simulations have not been used to any great extent and instead emphasis has been placed on the use of long-established techniques from conventional animation. Details of these facilities are given below.

5.3.1 The mass of an object

We described the method that Controller uses to take the mass of an object into account in section 3.5. We will now consider how the use of this facility will influence the animator.

The animator emphasises the mass or size of an object by carefully timing the motion of that object. He will make objects move more slowly as they get heavier, and perhaps give them more difficulty in controlling their bulk. When only a few objects are involved in a scene it is straightforward to time motion in this way using Controller's standard method of motion specification. The animator has to ensure that heavier objects cover less distance than lighter objects in the same period of time. This involves keeping track of the relative distances covered by objects of different mass so that new objects can be fitted in accordingly. If the scene is to contain many objects, however, the animator has to keep track of a large volume of information and his task becomes difficult. A method of calculating the motion of objects so that their mass is taken into account by Controller rather than by the animator was therefore provided.

To invoke Controller's mass facility the animator must define a mass value for each member of his cast. He is given the opportunity to enter this value at the keyboard whenever he selects a new cast member. The three 'ducks' illustrated in fig. 5.1., for example, have been defined so that the largest is also the heaviest

and the smallest is also the lightest. The unit of mass is immaterial as we are only interested in the relative motion of objects. In Controller the animator can select any value greater than zero to represent an object's mass. The selected value, however, is nearly always between one and one hundred and will rarely reach the thousands.

In our example (fig. 5.1), the animator has specified that each duck moves from rest with the same amount of acceleration. He has to define their track through the set and the duration of this acceleration phase, but Controller then takes over. We can see that after a slow start, where it appears that the ducks are overcoming their inertia, the distance attained by them is proportional to their mass. Note that it is the animator who usually defines the distance to be covered during a motion segment. The purpose of this exercise, however, is to save the animator from having to remember this information when a large number of objects are involved. He is therefore no longer required to specify distances in this case. We do not feel that this approach means that the animator suffers a significant reduction in his control over the motion specification. By selecting the value of the mass to be used he is still in general command of the situation.

5.3.2 Wind

Our next experiment was to determine whether wind could be incorporated into Controller. If the animator requires such an effect he must first define the strength of the wind and the direction that it is coming from. Wind strength is often indicated by a force number taken from the Beaufort wind scale. The following table summarises the Beaufort scale for land and will be familiar to most people (note that the relationship between the force number and wind speed is non-linear):

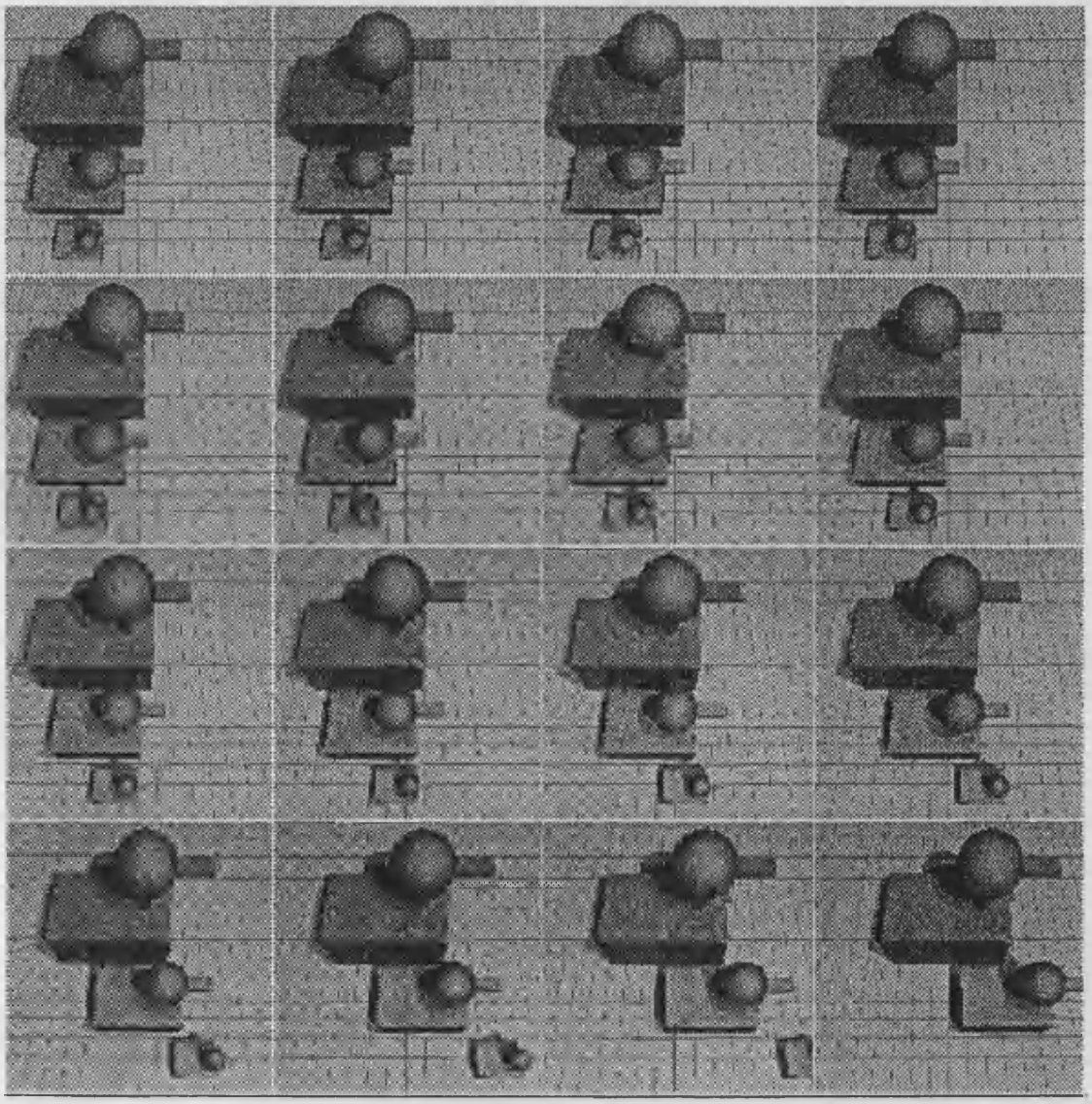


Figure 5.1. Motion of 'Ducks' with Different Mass

Beaufort Wind Scale		
Force	m.p.h	
0	1	light
1	1-3	light
2	4-7	light
3	8-12	gentle
4	13-18	moderate
5	19-24	fresh
6	25-31	strong
7	32-38	strong
8	39-46	gale
9	47-54	gale
10	55-63	storm
11	64-72	violent storm
12	>72	hurricane

In accordance with the Beaufort scale, we let the animator specify the force of the wind as a number between zero and twelve. The valuator used for inputing the intensity of a light is easily adapted to obtain this force value as well. Similarly the dial valuator is utilised to obtain the wind direction. These parameters are input before the animator defines the motion for any of the cast and they remain constant throughout the scene. To obtain effects such as gusting this will need to be extended so that wind force and direction can be varied over time. Such variation and lack of consistency add to the realistic animation of a wind.

Wind cannot be seen so it is modelled by the effects that it has on an object. Soft objects are not yet available in Controller and so this experiment is initially concerned with the effect of wind on rigid bodies. We identify two ways in which wind can affect such an object:

(i) making the object tilt or sway;

(ii) displacement of the object.

Making the object tilt in the direction in which the wind is blowing is an application of the traditional principle of anticipation. It gives an indication that the object may be about to move in the wind. Any displacement required is faired so that it appears more realistic. Our strategy is to add these wind effects onto the animator's normal motion definition.

In many cases wind effects are negligible and cannot be discerned by the viewer. A gentle breeze, for example, has no noticeable affect on a massive granite boulder. We therefore developed a quick test to determine if it is worthwhile calculating the tilt and displacement owing to the wind. We define the wind strength, S, to be

$$S = 2^B - 1 \qquad (*)$$

where B is the Beaufort scale force number. If the value returned by (*) is less than the mass of an object then we conclude that the wind has no noticeable affect on that object. These values can be precalculated and will be one of

0, 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, and 4095.

They are of the same magnitude as the mass values that the animator defines when using Controller (see §5.3.1). Comparing the mass of the object with the appropriate value obtained from (*) is then a straightforward task. A hurricane will therefore affect all objects with a mass less than 4095 units whereas a moderate wind affects only those objects with a mass less than 15 units. Many criteria could be used instead of (*) but we found this to be a reasonable method. It also reflects the non-linear relationship between the Beaufort force number and the wind speed.

We next need to calculate what the wind tilt and displacement will be. The maximum tilt angle reached will increase with the strength of the wind but decrease with the mass of the object. This relationship can be represented as

$$tilt\ angle = k \times \frac{wind\ strength}{mass}, \quad k\ constant.$$

For simplicity we use $k=1$. The tilt angle obtained from this formula tends to be greater than what would occur in reality. This means that the amount of tilt is exaggerated and is thus conforming to another of the traditional principles. How quickly the tilt angle is reached depends on the strength of the wind, slowly for a light wind but fast for a hurricane. Controller uses as a default value

$$number\ of\ interpolation\ frames = 2(13-B),$$

where B is the Beaufort scale force number. This formula returns a value between two and twenty four frames but there is no reason it should be imposed on the animator. He is allowed to assign his own value for the number of frames to be used if he desires. The interpolation of the tilt angle is calculated using equation (3.4a) so that the motion is faired. At the same time as tilting an object, the wind will begin to displace it. We know the force of the wind and the mass of the object so Newton's second law of motion is used to calculate the acceleration owing to the wind. The law of constant acceleration that we used previously (equation 3.4h) then supplies us with the wind displacement. As with the tilt, this displacement is applied in the direction of the wind. The resulting transformation to be administered to the object is placed into a *wind matrix*. This matrix is then applied after the animator has specified any other motion for the object. Figure 5.2 illustrates how the resulting object path is presented to the animator.

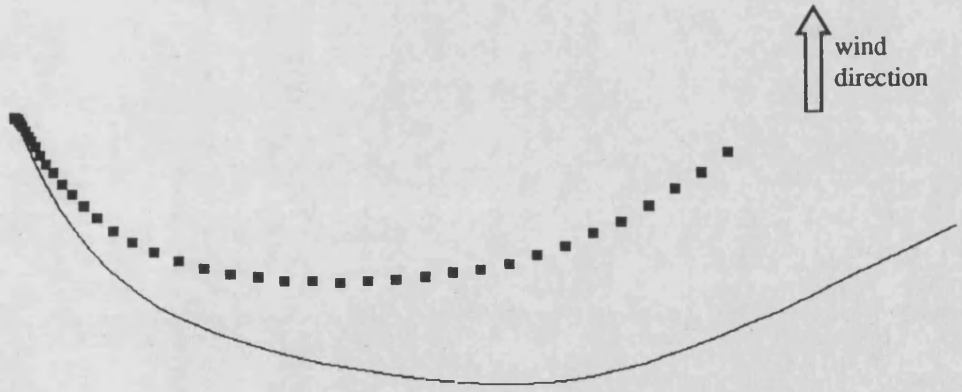


Figure 5.2. The Effect of Wind on a Motion Path

Here the animator has selected a wind of force five and given the object a mass of twenty five units. After the track is drawn, the object is accelerated from rest and then maintained at a constant speed. Note how the displacement of the object from the track increases with successive frames.

In the example of fig. 5.3 an otherwise stationary 'clown' character has been exposed to a wind simulated by the animator. If viewed at the appropriate rate, these frames provide a good demonstration of how convincing the wind facility can be. Several factors still need to be considered, however. The animator will not want the wind to affect fixed objects such as buildings and so will require the option of defining objects to be immune from the wind. In our current implementation, assigning an object a mass of greater than 4095 units achieves this. Also, if the object tilts so much that it becomes unstable then we have to provide some mechanism for it to topple over. At the moment it is up to the animator to refine such motion until it appears visually correct. So far we have assumed that the cast do not attempt to counteract the effects of a wind but this is not always the case. Human characters, for example, will often tilt into the wind (to obtain greater stability as they move) rather than being tilted away from it. The displacement of an object away from the track drawn for it by the animator can also present problems. The object may be 'blown' into other objects or even disappear from the set. The animator, however, can always redefine the object path or wind parameters to overcome these problems.

5.3.3 Staggers

A *stagger* is the term applied to the animation of an action involving a vibration or oscillation (White 1986). A javelin hitting the ground, for example, will usually vibrate after impact. Its animation will consist of a series of oscillations gradually decreasing in amplitude until an equilibrium position is reached. A character can also be made to shake with fear by the appropriate application of a stagger. A stagger is therefore a useful tool for the traditional animator and we decided to provide a similar facility in Controller.

The physical model of a stagger would be based on a damped oscillation. For example, Smyrl (1978) states the formula for a slightly damped oscillation as

$$s = e^{-\rho t} R \cos\{(\omega^2-\rho^2)^{\frac{1}{2}} t-\alpha\},$$

where s is the displacement, t is the time, R is the amplitude of the oscillation, α

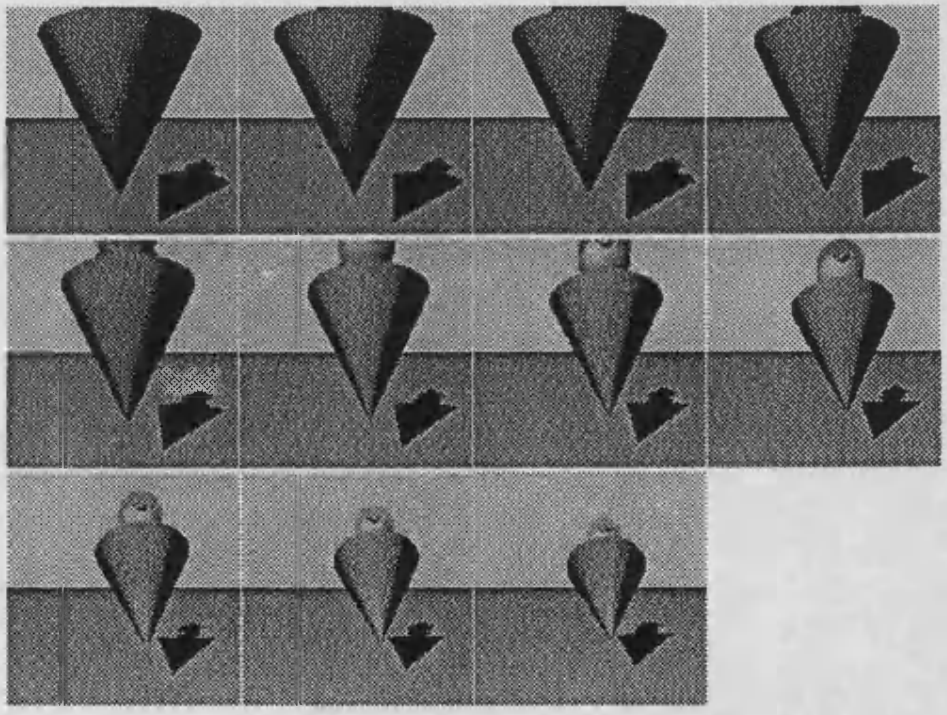


Figure 5.3. The Effect of a Wind on the 'Clown' Character

is the phase angle, and ρ and ω are constants. This equation is unlikely to be familiar to an animator and would be difficult for him to use effectively. An alternative approach is to base the computer version of a stagger on the traditional procedure. The animator is then in familiar territory and he can use his skill to the best effect. We have preferred to use this second approach.

To define a stagger in Controller, the animator is required to:

(i) select the frame positions between which the stagger is to occur;

(ii) enter the number of oscillations required at the keyboard;

(iii) use the dial valuator to enter the angle of the object's maximum swing from its equilibrium position.

The amplitude reached on successive oscillations will gradually decay until the object comes to rest at its equilibrium position. This decay can be carried out linearly or by using some more complicated interpolation method, according to the animator's wishes. We usually find that a simple linear division produces acceptable results. The time taken for each oscillation must also be allocated. As the object ends at rest and the amplitude of its extreme positions is decaying, fewer frames are needed for each successive oscillation. The deceleration function (3.4b) is applied to the total number of frames defined for the stagger to obtain the actual frame allocation for each oscillation. The final step is to interpolate the oscillation between the object's current extreme position and its next extreme position. If it takes longer to come out of an extreme than it does to enter it then the resulting motion will appear to have more 'snap'. To exaggerate the oscillations in this way they are interpolated using the acceleration function (3.4a) over the interval $[0,\pi/2]$. Figure 5.4 illustrates an example of a stagger obtained from Controller. Here a child's toy has been 'pushed' from its equilibrium position and oscillates four times. Such a stagger is quickly calculated using the technique that we have described above and is much cheaper to evaluate than the damped oscillation formula is.

The use of staggers in Controller presents new possibilities for their application. For example, when a heavy object falls to the ground the animator

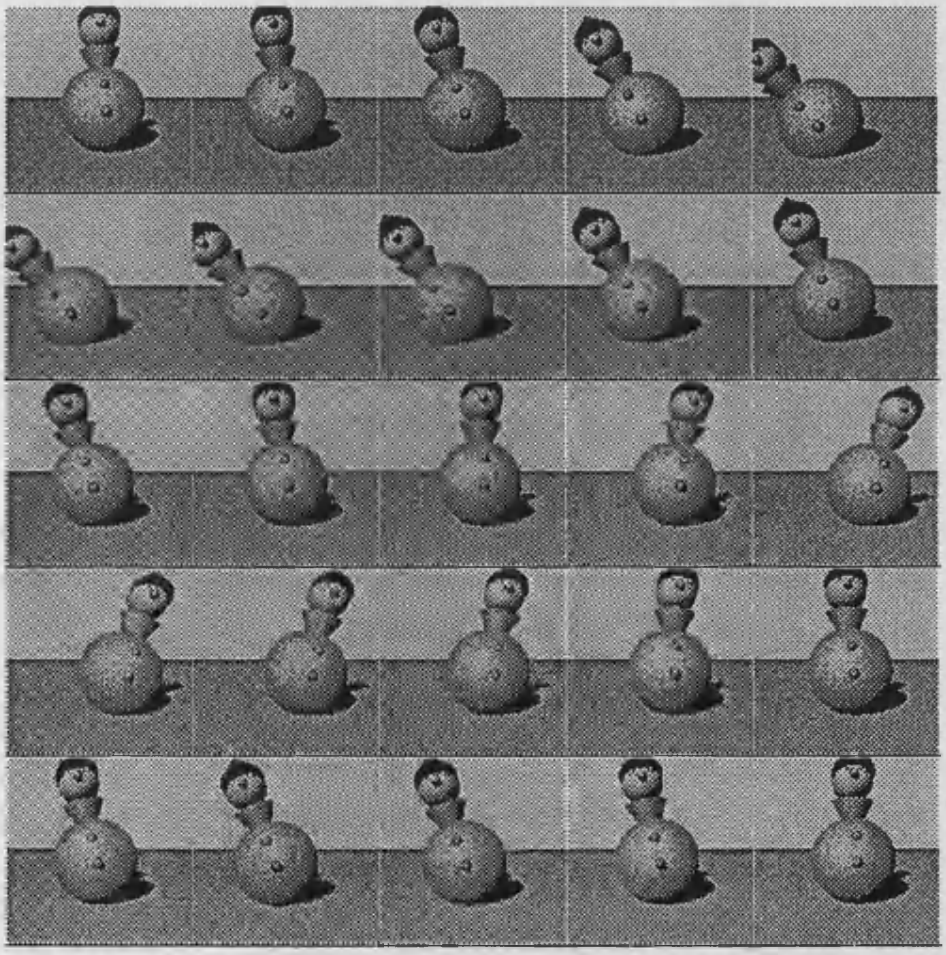


Figure 5.4. A Child's Toy After it has Been Made to Stagger

will often require the ground to tremor as a result of the impact. Applying a stagger to the view point of a virtual camera will give this impression. The implementation of a stagger in Controller is not yet complete, however. At present we restrict the animator to align an object's equilibrium position with a principle axis from its own coordinate space. We will have to extend this so that a stagger can occur in any direction. How the mass of an object affects a stagger could also be considered.

5.3.4 Gravity

An object released from above the earth's surface will fall towards it with an ever increasing velocity. The rate of change of this velocity is called the acceleration due to gravity, g, and is the same for all objects at a given location on the earth. The actual value of g varies between $9.78ms^{-2}$ and $9.83ms^{-2}$, depending on the location. Gravity also causes an ascending object to decelerate until its velocity reaches zero, after which point it begins to fall. An animator will often want to reflect the effects of gravity in an animation sequence. Once again, Controller will assist him in this carrying out task.

The animator defines the height of an object at its start and end frames and Controller calculates the inbetweens. The constant acceleration law (3.4h) is appropriate for modeling gravity. Here, however, we do not calculate the acceleration required to precisely satisfy the displacement and time constraints of the motion segment. To mimic gravity, the same value must be used for the acceleration due to gravity on all objects. This could be the value of g quoted above even though Controller's environment is artificial and does not use the metric system of measurement. The only criterion that must be satisfied in Controller is that the motion obtained is visually realistic. To simplify the inbetween calculations we use a value of 10 units for g and find that this produces acceptable results. The animator can also specify his own value for g as he may, for example, be setting the animation in outer space.

The level of the ground plane is obtained from the set model currently being used. When a falling object reaches this level it will either carry on travelling

below the surface of the ground, or come to an abrupt halt. Usually the animator will require the object to be stopped by the ground. An example of such motion is depicted in fig. 5.5.

Figure 5.5 An Object Falling Under Gravity

Here the object is falling from rest and is accelerating towards the ground. The distance travelled between the final two frame positions, however, is less than the preceding inter frame distance. This gives the effect that the object is 'crashing' into the ground. The animator can enhance the impact by making the object bounce or by applying a stagger to the camera. Note that if the impact occurs before the time specified for the motion segment has elapsed then any residual frames are ignored.

An animator will not want a feather to fall to the ground as quickly as a cannon ball. A feather is subject to more air resistance than the cannon ball and consequently takes longer to fall. Similarly, its ascent is retarded by air resistance. Controller therefore allows the animator to take air resistance into account. He assigns each object a value, a_r, between 0 and 1. When $a_r = 0$ there is no air resistance and when $a_r = 1$ air resistance will completely counteract gravity. This value is incorporated into the calculation by scaling the interpolated height by $(1-a_r)$. A pictorial example of the effects of air resistance is given in fig. 5.6 that depicts two spheres rising and falling under gravity. For the sphere

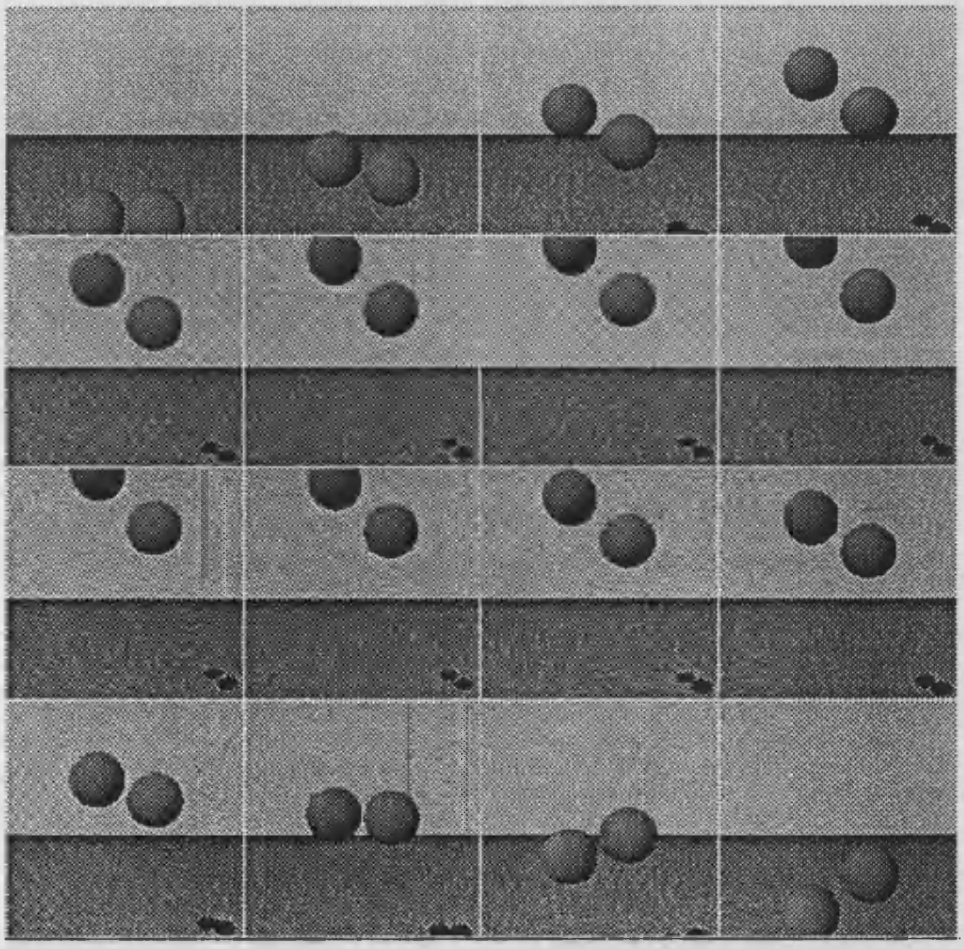


Figure 5.6. Gravity and Air Resistance

on the left $a_r = 0$, whilst for the sphere on the right $a_r = 0.5$.

5.4 Controller in Use

The main purpose of Controller is to allow us to experiment with different methods of defining computer animation. We aim to provide a flexible system that does not overlook the craft of the traditional animator, and have developed Controller accordingly. A practical test will help show our success in achieving this aim. A student* from an arts background was therefore asked to use the current version of our system to generate a computer animation sequence. This section details the strengths and weaknesses of Controller that became apparent during the making of this sequence.

5.4.1 Preliminaries

The student (hereafter referred to as the animator) decided to produce a commercial for his college degree show. As with all animation production, the first step is to write the script and a synopsis of this is given below. To present the action in more detail the animator also prepared a storyboard and an extract from this can be found in Appendix C.

The sequence begins with a camera shot of a dimly lit room. A large window covered by a Venetian blind is on the far wall, and a large television monitor is mounted on a side wall. On the floor of the room there are seven dustbin-like wheeled objects each painted in a different colour. An alarm clock hanging on the wall above the window suddenly begins to flash and appears to have an effect on the 'dustbins'. It turns out that they are a group of robots resting in their dormant state. The alarm has roused them, however, and their eyes gradually illuminate whilst their heads and wheels extend out from their bodies. Once all the robots have been activated the chief robot decides to inspect his subordinates. The last to be inspected is a blue robot but while he is under

* Peter Wong Ming, Graphics Design student, Bath College of Higher Education.

the gaze of his chief he topples over and crashes to the floor. The chief shakes his head in dismay and realises that the blue robot is in need of major repair. At this point the television monitor energises and displays an advert for a degree show. The robots have to hurry if they are to make the show in time and so the chief leads them out of the room. Meanwhile the blue robot has managed to get up off the floor but his head has been dislocated by the fall. This injury has affected his coordination and although he attempts to follow the others, his movement is erratic.

The scene now changes to a strange alien landscape dominated by a huge dome. This is the location of the degree show and six of the robots can be seen making their way inside. They are followed by various other visitors to the show but there is no sign yet of the faulty robot. By the time he does appear the degree show is over and the other robots are leaving the dome and heading back home. They do not notice their compatriot and he is knocked over by one of them as they pass. The faulty robot attempts to get up but this time he does not succeed. He turns his head towards the camera and he looks appealingly at the audience as the scene fades out.

5.4.2 Generating the Models

The information contained in the script and storyboard enable the models required for the animation to be generated. The animator is not expected to do this, however, as Controller does not yet include an interactive modeling facility. We have to prepare cast and set models using the procedure described earlier (§2.3).

Two sets are required, one for the room where we first meet the robots and another for the alien landscape. Both of these contain objects that are easily built out of quadrics and so are straightforward to generate. Cast models are then needed for the robots and the other visitors to the degree show and are again built out of quadrics. The cast will often exhibit some form of internal motion as well. An activated robot, for example, keeps his head fixed but swings his body with a slight oscillatory action. We describe such motion by using an appropriate set of poses (see §2.3.3), here we have used those depicted in fig. 5.7. A model

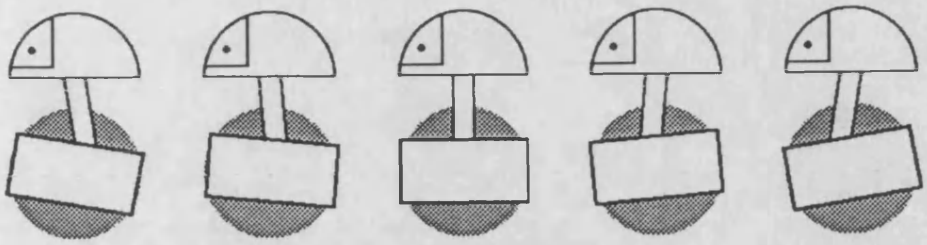


Figure 5.7. Motion Poses for a Robot Character

containing only a robot head and another model containing only a robot body were also provided. These will make it possible for the animator to have more control over the head and rotate it in a different direction from the rest of the body (see §5.4.3). A colour plate showing the cast used for the animation sequence can be found in Appendix B.

5.4.3 Defining the Animation

Once the models have been generated, careful planning or *staging* of the animation is necessary. A detailed description of how the animator carried out his plans is not needed here. We will concentrate on the occasions when a particular strength or weakness of Controller became evident.

(i) Activating the robots

At the beginning of the sequence the robots are transformed from their dormant state to their active state. This process was straightforward to carry out by using Controller's motion pose technique. The robot activation is a mechanical process and so was modelled effectively by this method. The motions of the head and wheels away from the body were interpolated using equation (3.4a) over $[0,\pi]$ and thus appeared visually smooth. The head motion is depicted in fig. 5.8. The

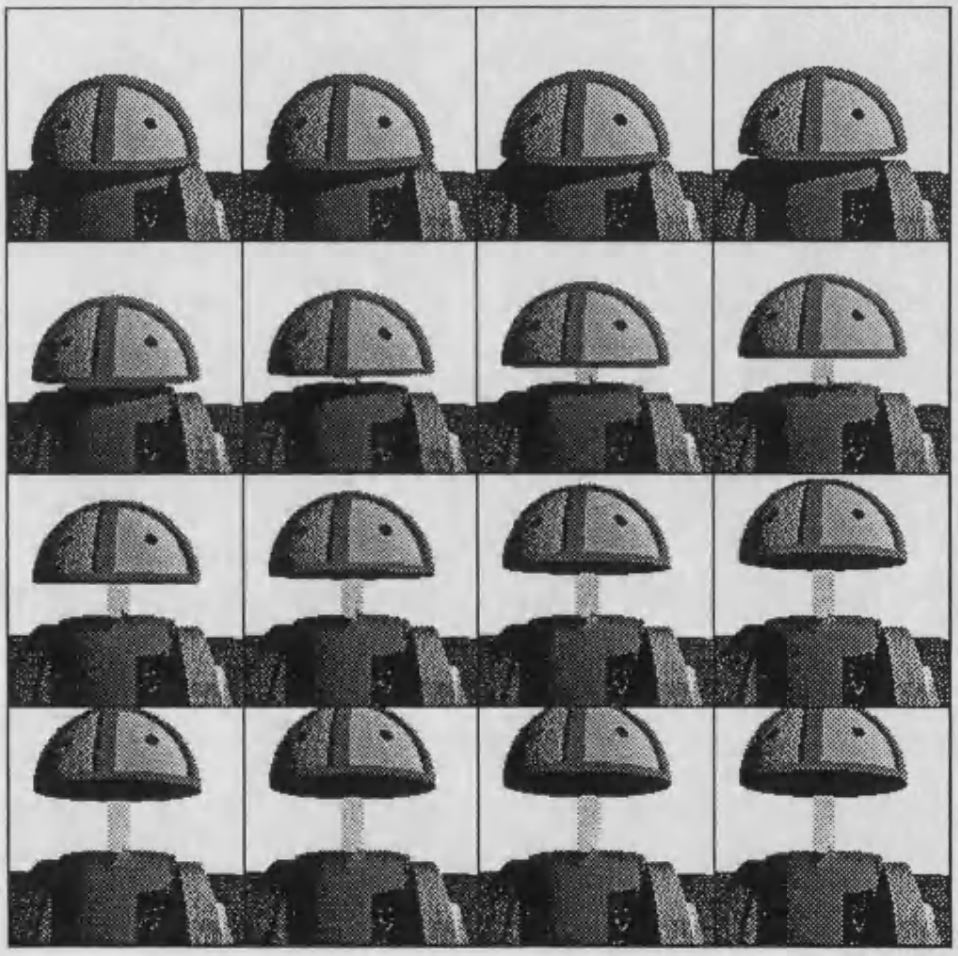


Figure 5.8. A Robot's Head Motion During its Activation

colour of the robots' eyes were also interpolated from black (when dormant) to white (when activated) using the same method.

(ii) Plotting the path of a single object

Controller's method of path planning was used effectively by the animator when defining the motion of a single object. The control that he had over both the spatial and temporal aspects of the path allowed him to produce the exact motion he wanted. The B spline editor, undo facility, and pencil test proved particularly useful to him. Most of the motion in the final sequence was refined and smoothed using the facilities offered by Controller.

(iii) Moving the cast in different directions

When many objects move at the same time and in different directions, care must be taken to ensure that they do not collide with one another. The animator had to consider this when all the robots left for the degree show (fig. 5.9). Using Controller a visual check determined if the cast would collide. The proximity of the motion paths was one guide as to whether this would occur but these paths did not give an impression of the size of the character in the scene. Therefore possible collisions might have been missed. Another method that proved useful here was to generate a wire frame sequence of the action. Such a sequence was fast to generate and identified several points in the action where adjustments were needed. Some form of automatic collision detection in Controller would be desirable, however, but has yet to be carried out.

(iv) Moving the cast in a procession

When the robots enter the dome they do so in a procession, that is, in single file one behind the other. Although the animator could define each object's path individually to produce such a procession, this is unnecessarily tedious. We therefore provided a new facility that enabled the animator to use the same motion path for several objects with a specified time lag between the original object and the current object. A still from the robot sequence demonstrating the

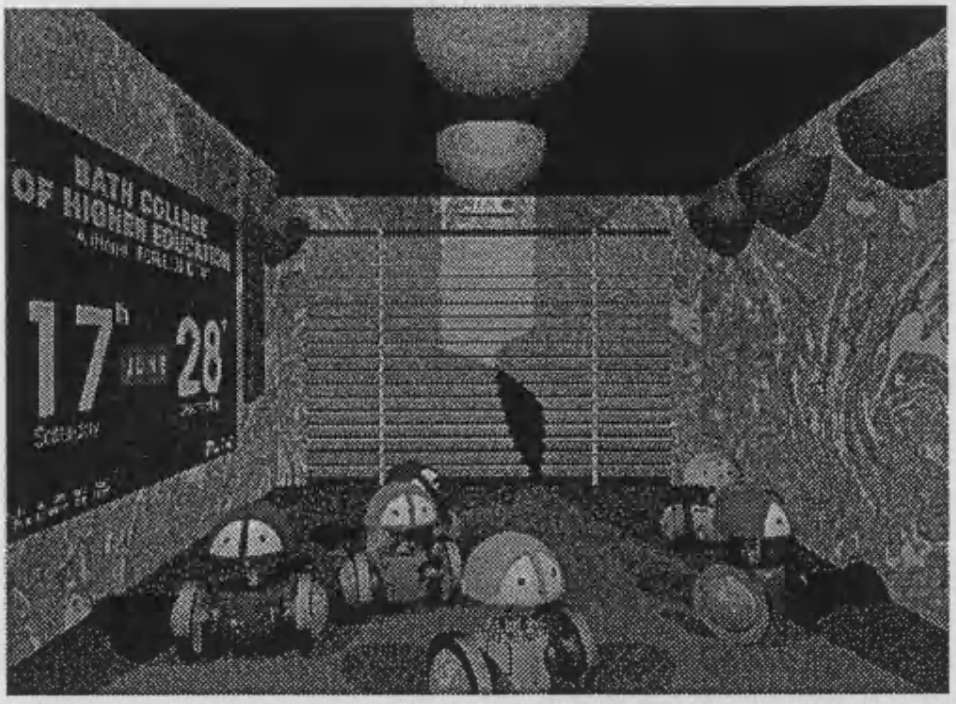


Figure 5.9. The Robots Leave for the Degree Show

robot procession is given in fig. 5.10. Note that there is a gap in this procession. The animator has added more interest (or *appeal*) to the action by making a robot break out of the procession and rush into the dome ahead of the others.

(v) A falling robot

The faulty robot falls to the ground with a crash at two points in the animation sequence. The stagger facility described earlier in this chapter (§5.3.3) proved useful here. Applied to the view point of the camera after the robot hits the ground, the stagger gave the impression that the ground was shaking.

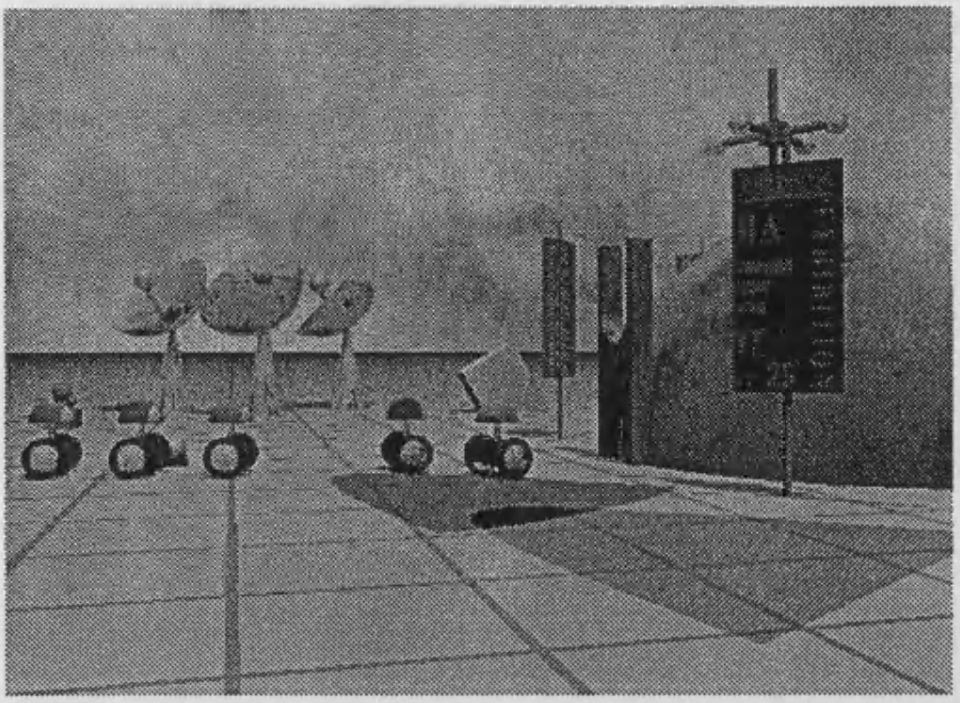


Figure 5.10. The Robots Arrive at the Dome

(vi) Head movements

After the faulty robot falls over for the first time, the chief shakes his head in dismay and then turns his head towards the television monitor. So that the animator can manipulate the head in this way, separate models of the robot's head and body were provided (fig. 5.11). The facility that allows a path to be occupied by more than one member of the cast was again useful here. The animator began by defining the path for the robot's body. The head was then defined to occupy the same path but without any time lag. As long as the head is maintained at the correct height then the resulting robot appears no different from the others. Now, however, the animator can set the orientation of the head independently from its body and achieve the desired head movements.

The alternative to this approach is to use an articulated model of the robot

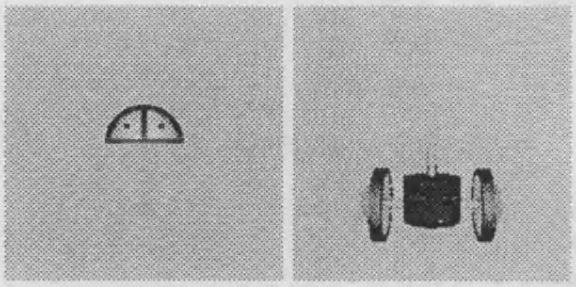


Figure 5.11. Models of a Robot's Head and Torso

but a more complex interface is then needed to manipulate it. The animator has to ensure that a movement is applied to the correct part of the model hierarchy so that only the required components of the robot are affected. In our method the animator can consider the head as a separate entity that he has defined to follow the same track as the body. The body does not then have to be considered while he is manipulating the head. We feel that this provides the animator with a more natural way of defining the movements of the robot head and the results are just as effective.

(vii) Holds

The cheapest way of generating a hold or pause in the action is to display the same frame for some specified length of time. This technique was used in the robot sequence but there are drawbacks. The resulting hold is stiff and frozen and will often look unnatural. This occurred when the satellite dishes found on the alien landscape suddenly stopped rotating owing to the specification of such a pause. Even during a hold it looks far better if some motion continues to flow. The satellite dishes should therefore have continued to rotate when the other characters halted. The animator could have easily arranged for this to happen in Controller but of course many more frames would then have needed to be rendered. Owing to the animator's deadline for completion, he opted for the least expensive method.

(viii) Camera movements

Controller's facilities for manipulating a camera were used extensively by the animator. In the robot animation sequence tracks, zooms and pans were put to good effect. At one point in the action the animator required the view point of the camera to be inside the chief robot. We therefore allowed a path already defined for a cast member to also be occupied by a camera. This enabled the camera to exactly mimic the overall motion of the chief robot and so gave the desired effect.

(ix) Lighting

Providing effective illumination of a scene is a difficult task for the animator. Using Controller he can position lights about a set and define their intensity. The problem is that we do not know the extent of the illumination provided by them until an example image has been rendered using the ray tracer. Often the resulting scene is far too dark. We found that the best approach was to start with a uniform even illumination of the set using light locations and intensities determined at the modeling stage. If desired, the animator can then use Controller to vary the default lighting.

5.4.4 The Final Touches

Having specified the data required to produce the robot animation, the individual frames could be rendered. A wire frame version of each action was first generated and viewed using the **video** facility (§4.6). This highlighted the adjustments that needed to be made before the expensive ray tracer was invoked. Controller can easily be reactivated at this stage to make the necessary changes.

The **video** facility is also useful for viewing a sequence that is *double framed.* Double framing an action is a common technique for reducing the cost of animation production. It involves using each frame twice so that the action can be depicted in only half the number of frames that would otherwise be required. Except for rapid movements, the human eye cannot detect whether an action has

been double or single framed. In the robot sequence, for example, every action except for the camera stagger (staggers are more effective when they occur quickly) was double framed. Note that it is the animator and not Controller who has to define a motion path according to whether double or single framing is to be used. It would be straightforward, however, to enhance Controller so that if an action is to be doubled frame then only alternate frame positions are calculated.

The frames were then rendered using the ray tracer (see colour plates 5 and 6) and transferred onto video tape. Finally, the animation was completed by adding a soundtrack composed by the animator. A local television company* provided us with the facilities to apply the required music and sound effects onto the video tape.

5.5 Summary and Conclusion

The methods described at the beginning this chapter show some of the ways in which traditional principles can be used in computer animation. They enable visually realistic animation to be generated quickly and cheaply. We do not want to build into Controller too many facilities that will have infrequent use, however (for example, the wind facility). Our aim is to provide the animator with a tool box of animation techniques that can be combined in any manner. He will then be able to carry out any animation effect that he desires. The examples given in this chapter are meant to demonstrate how this can be done.

The way in which Controller is used to generate the robot sequence also reinforces our claim that traditional skills should not be overlooked. The student often exaggerated the action, added appeal, faired the motion, and so on. Controller is being developed so that an animator can apply his craft in the majority of cases. The production of the robot animation also helps us to assess how usable Controller is at present. After an initial period of familiarisation with the system we found that the student soon got an idea of its capabilities.

* HTV West, Bristol.

Although he needed periodic assistance throughout the production of the animation, we have seen that he was able to effectively utilise many of Controller's facilities. We were also able to identify and implement new facilities for our system. We feel that the results of this experience augur well for the future of Controller.

Chapter 6

Conclusion

6.1 Introduction

At the beginning of this thesis we set out to:

(i) provide a system of animation planning that is straightforward to use but at the same time keeps the animator in control;

(ii) enhance the animation effects attainable by incorporating the principles of traditional animation.

We have been interested in producing animation for entertainment (rather than scientific) applications and have developed techniques of planning animation accordingly. In this final chapter we will consider how successful we have been in satisfying the above aims and propose possible areas for future investigation.

6.2 Summary of Results

We can identify several areas where the work carried out during the thesis has been worthwhile and these are summarised below.

6.2.1 Planning Animation

The *Controller* system for planning three dimensional computer animation has been developed. At all stages of its operation Controller is intended to be straightforward and flexible to use. To contribute to the success of this goal we have:

- developed Controller as an interactive graphics system so that the animator is provided with an immediate visual response;
- used kinematic techniques (rather than more complicated strategies) for modeling motion;
- allowed the animator to undo or alter his specification at will;
- kept the implementation of Controller as simple as possible so that the need for a complex interface is avoided.

We have experimented with a variety of methods that can be used to specify an animation and made the best of these available in Controller. The animator should not be offered too many facilities that he will rarely (if ever) use, however. Instead we provide him with the necessary tools to perform his desired task and let his skill determine the effectiveness of the results. The animator, and not the animation system, controls the quality of the animation produced.

6.2.2 Adaptation of Conventional Animation Techniques

We have modelled motion by using uncomplicated kinematic methods that are straightforward to use and can be evaluated quickly. At the same time, however, we wanted to ensure that convincing animation could be produced. We found that an effective approach was to make use of the principles of animation developed by conventional animators. For example, an animator using Controller can:

- use arcs when drawing tracks and so avoid motion that is mechanical in appearance;
- time the motion of several objects to mimic the effects of mass;
- slow in and slow out the motion of an object both along its path and between an interpolation;
- specify the extremes of an interpolation so that the resulting motion is exaggerated.

The application of these techniques result in animation that is convincing in an artistic rather than in a scientific sense and so is ideal for animation aimed at

the entertainment market. We did not have to adhere to computationally expensive physical laws to implement them and the animator is provided with the familiar tools of his trade.

6.2.3 The Portability of Controller

Our approach of 'faking reality' is cheaper to implement than a full dynamic implementation and motion interpolations are fast to calculate. Controller has also been written in the highly portable C programming language. These factors make it feasible for Controller to be installed on machines where the computing power available is limited (such as portable computers). Our system for animation planning could therefore be made more widely available and offer many animators the chance to experiment with the computer medium. The final animation might still have to be rendered on a more powerful machine, however, especially if realistic shaded images are required. The animation data then has to be transferred to an appropriate host but this procedure should not present any major problems.

6.3 Future Investigation

There are several areas where the techniques of planning animation developed during the thesis could be improved and new techniques evolved.

6.3.1 Further use of the Traditional Principles of Animation

The use we have made of traditional animation principles have produced promising results and there is scope here for further investigation. Several of the principles described in chapter 1 have not yet been considered. Squash and stretch, for example, should be provided so that the rigidity of moving objects is reduced. Such an effect would be carried out at the rendering stage by applying the appropriate deformations to the object. Within Controller we will need some mechanism whereby the animator can define the amount of deformation required and at what points in the action it should occur.

Controller is a key frame animation system and so it generates animation from pose to pose. The alternative is to animate 'straight ahead', a method that introduces more spontaneity into the action. The motion path technique of planning animation is unsuitable for this purpose. Here the animator defines each frame in sequence from the beginning of the scene to its end. To carry out this procedure without some form of frame interpolation would be laborious, however. Determining the best way of providing for this would be an interesting area to develop.

Another possibility is to provide automatic anticipation of an action before it is initiated. The animator would define the required action as usual but also have the option of requesting it to be anticipated. The animation system would then begin by calculating the 'opposite' of the defined motion over some specified number of frames. For example, if the animator defines an object to move quickly to the right of the set then this action could be anticipated by first moving the object (in reverse) to the left of the set. Such a 'wind-up' anticipation is often found in animated cartoons when a character is chased off the scene (White 1986). Assisting the animator with follow through, overlapping action, and secondary action could also be investigated.

6.3.2 Improvements to Controller

The prototype version of Controller is not without its flaws and several areas could be improved. Defining the orientation of a cast member, for example, would be aided by displaying a line drawing of the object whilst it is being manipulated into different configurations. We also need to develop the way in which the internal motion of the cast is modelled. One possibility is to use articulated objects but this will also complicate the animator's task of planning the movements of such objects. Alternatively, the technique that we described in chapter 5 for allowing several objects to share the same motion path could be extended further. Note that the motion pose method need not be discarded altogether as it is an effective and simple way of modeling repetitive motions.

The interpolation methods offered could also be improved and new ones developed. For example, our automatic mass facility has only been used during the definition of a motion path. Mass could be considered at the parametric animation stage as well. Again, however, such a facility is only needed when there are many cast members involved in a scene. We have not yet investigated how the family of acceleration curves formula can be used to model the forces applied to objects (see chapter 3).

Other possibilities include varying the direction in which motion paths can be traversed, and the provision of a facility to coordinate the animation with its soundtrack.

References

Armstrong, W.W. and Green, M. W. (1985). The Dynamics of Articulated Rigid Bodies for Purposes of Animation. *The Visual Computer 1*, pp. 231-240.

Armstrong, W.W. Green, M. and Lake, R. (1986). Near-Real-Time Control of Human Figure Models. *Proc Graphics Interface '86*, pp. 147-151.

Arnaldi, B. Dumont, G. Hegron, G. Magnenat-Thalmann, N. Thalmann D. (1989). Animation Control with Dynamics. *Proceedings of Computer Animation '89*, pp. 113-123.

Arya K. (1986). A Functional Approach to Animation. *Computer Graphics Forum 5*, pp. 297-312.

Badler, N.I. O'Rourke, J. and Toltzis, H. (1979). A Spherical Representation of a Human Body for Visualizing Movement. *Proc. of the IEEE 67(10)*, pp. 1397-1403.

Badler, N.I. Smoliar, S.W. (1979). Digital Representations of Human Movement. *Computing Surveys 11(1)*, pp. 19-38.

Badler, N.I. (1989). Artificial Intelligence, Natural Language, and Simulation for Human Animation. *Proceedings of Computer Animation '89*, pp. 19-31.

Baeker, R. M. (1969). Picture-Driven Animation. *Proc. Spring Joint Computer Conference*, pp. 273-288.

Van Baerle, S. (1987). Combining Computer Graphics and Traditional Animation. *Eurographics 87 Character Animation Tutorial*, pp. 134-145.

Barr, A. H. (1984). Global and Local Deformations of Solid Primitives. *Proc. SIGGRAPH '84. Computer Graphics 18(3)*, pp. 21-30.

Barsky, B.A. and Greenberg, D.P. (1980). Determining a Set of B-Spline Control Vertices to Generate an Interpolating Surface. *Computer Graphics and Image Processing 14*, pp. 203-226.

Barzel, R. and Barr, A.H. (1988). A Modeling System Based On Dynamic Constraints. *Proc. SIGGRAPH '88. Computer Graphics 22(4)*, pp. 179-188.

Bethel, E.W. and Uselton, S.P. (1989). Shape Distortion in Computer-Assisted Keyframe Animation. *Proceedings of Computer Animation '89*, pp. 3-17.

Brotman, L.S. and Netravali, A.N. (1988). Motion Interpolation by Optimal Control. *Proc. SIGGRAPH '88. Computer Graphics 22(4)*, pp. 309-315.

Burtnyk, N. and Wein, M. (1971). Computer Generated Key-Frame Animation. *Journal of the SMPTE 80*, pp. 149-153.

Burtnyk, N. and Wein, M. (1976). Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation. *Communications of the ACM 19(10)*, pp. 564-569.

Calvert, T.W. Chapman, J. and Patla, A. (1980). The Integration of Subjective and Objective Data in the Animation of Human Movement. *Proc. SIGGRAPH '80. Computer Graphics 14(3)*, pp. 198-203.

Catmull, E. (1978). The Problems of Computer-Assisted Animation. *Proc. SIGGRAPH '78. Computer Graphics 12(3)*, pp. 348-353.

Catmull, E. (1979). New Frontiers in Computer Animation. *American Cinematographer*, pp. 1000-1053.

Chuang, R. and Entis, G. (1983). 3-D Shaded Computer Animation - Step by Step. *IEEE Computer Graphics and Applications, December*, pp. 18-25.

Cook, R.L. (1986). Stochastic Sampling in Computer Graphics. *ACM Transactions on Graphics 5(1)*, pp. 51-72.

Denber, M.J and Turner, P.M. (1986). A Differential Compiler for Computer Animation. *Proc. SIGGRAPH '88. Computer Graphics 20(4)*, pp. 21-27.

Entis, G. (1986). Computer Animation - 3D Motion Specification and Control. *SIGGRAPH '86 Tutorial Notes.*

Foley, J.D. and Van Dam, A. (1982). *Fundamentals of Interactive Computer Graphics.* Addison Wesley.

Forest, L. Magnenat-Thalmann, N. and Thalmann, D.(1986). Integrating Key-Frame Animation and Algorithmic Animation of Articulated Bodies. *Proc. of Computer Graphics Tokyo '86*, pp. 263-273.

Forest, L. Rambaud, R. Magnenat-Thalmann, N. and Thalmann, D. (1986b). Keyframe-Based Subactors. *Proc Graphics Interface '86*, pp. 213-216.

Glassner, A.S. (1988). Spacetime Ray Tracing for Animation. *IEEE Computer Graphics and Applications, March*, pp. 60-70.

Gomez, J.E. (1984). Twixt: A 3D Animation System. *Proc. Eurographics '84*, pp. 121-133.

Grosso, M.R. Quach, R.D. and Badler, N.I.(1989). Anthropometry for Computer Animated Human Figures. *Proceedings of Computer Animation '89*, pp. 83-95.

Hahn, J.K. (1988). Realistic Animation of Rigid Bodies. *Proc. SIGGRAPH '88. Computer Graphics 22(4)*, pp. 299-308.

Hanrahan, P. and Sturman, D. (1985). Interactive Animation of Parametric Models. *The Visual Computer 1*, pp. 260-266.

Herbison-Evans, D. (1978). NUDES 2: A Numeric Utility Displaying Ellipsoid Solids, Version 2. *Proc. SIGGRAPH '78. Computer Graphics 12(3)*, pp. 354-356.

Herbison-Evans, D. (1982). Real-Time Animation of Human Figure Drawings with Hidden Lines Omitted. *IEEE Computer Graphics and Applications, November*, pp. 27-33.

Hubschman, H. and Zucker, S.W. (1982). Frame-to-Frame Coherence and the Hidden Surface Computation: Constraints for a Convex World. *ACM Transactions on Graphics 1(2)*, pp. 129-162.

John, N.W. and Willis, P.J. (1989). The Controller Animation System. *Computer Graphics Forum 8(2)*, pp. 133-138.

John, N.W. and Willis, P.J. (1989b). Some Methods to Choreograph and Implement Motion in Computer Animation. *Proceedings of Computer Animation '89*, pp. 125-139.

Kallis, S.A. (1971). Computer Animation Techniques. *Journal of the SMPTE 80(3)*, pp. 145-148.

Kingslake, R. (1985). *An Introductory Course in Computer Graphics.* Chartwell-Bratt Studentlittetatur.

Kochanek, D.H.U and Bartels, R.H. (1984). Interpolating Splines with Local Tension, Continuity, and Bias Control. *Proc. SIGGRAPH '84. Computer Graphics 18(3)*, pp. 33-41.

Korein, J. and Badler, N.I. (1983). Temporal Anti-Aliasing in Computer Generated Animation. *Proc. SIGGRAPH '83. Computer Graphics 17(3)*, pp. 377-388.

Korein, J.U. and Badler, N.I. (1982). Techniques for Generating the Goal-Directed Motion of Articulated Structures. *IEEE Computer Graphics and Applications, November*, pp. 71-81.

Lansdown, J. (1983). Object and Movement Description Techniques for Animation: An Informal Review. *First Australian Conference on Computer Graphics*, pp. 82-85.

Lasseter, J. (1987). Principles of Traditional Animation Applied to 3D Computer Animation. *Proc. SIGGRAPH '87. Computer Graphics 21(4)*, pp. 35-44.

Lewell, J. (1985). Behind the Scenes at Hanna-Barbera. *Computer Pictures*, pp. 15-43.

Lundin, R.V. (1984). Motion Simulation. *Proc. NICOGRAPH '84*, pp. 2-10.

Madsen, R. (1970). *Animated Film.* Interland Publishing Inc.

Magnenat-Thalmann, N. and Thalmann, D. (1985). *Computer Animation Theory and Practice* Springer-Verlag.

Magnenat-Thalmann, N. and Thalmann, D. (1985b). Subactor Data Types as Hierarchical Procedural Models For Computer Animation. *Proc. Eurographics '85*, pp. 121-128.

Magnenat-Thalmann, N. and Thalmann, D. (1985c). Three-Dimensional Computer Animation: More an Evolution Than a Motion Problem. *IEEE Computer Graphics and Applications, October*, pp. 47-57.

Magnenat-Thalmann, N. Thalmann, D. and Fortin, M. (1985). Miranim: An Extensible Director-Orientated System for the Animation of Realistic Images. *IEEE Computer Graphics and Applications, March*, pp. 61-73.

Magnenat-Thalmann, N. and Thalmann, D. (1986). Special Cinematographic Effects With Virtual Movie Cameras. *IEEE Computer Graphics and Applications, April*, pp. 43-50.

Magnenat-Thalmann, N. and Thalmann, D. (1986b). Artificial Intelligence in Three-Dimensional Computer Animation. *Computer Graphics Forum 5*, pp. 341-348.

Magnenat-Thalmann, N. and Thalmann, D. (1988). Construction and Animation of a Synthetic Actress. *Proc. Eurographics '88*, pp. 55-66.

Magnenat-Thalmann, N. (1989). The Problematics of Facial Animation. *Proceedings of Computer Animation '89*, pp. 47-55.

Marino, G. Morasso, P. and Zaccaria, R. (1985). NEM: A Language for Animation of Actors and Objects. *Proc. Eurographics '84*, pp. 129-140.

Max, N. (1989). A 3-D Error Diffusion Dither Algorithm for Half-Tone Animation on Bitmap Screens. *Proceedings of Computer Animation '89*, pp. 169-179.

Miller, G.S.P. (1988). The Motion Dynamics of Snakes and Worms. *Proc. SIGGRAPH '88. Computer Graphics 22(4)*, pp. 169-178.

Millerson, G. (1973). *TV Camera Operation, Focal Press.*

Moore, M. and Wilhelms, J. (1988). Collision Detection and Response for Computer Animation. *Proc. SIGGRAPH '88. Computer Graphics 22(4)*, pp. 289-298.

Noma, T. and Kunii, T.L. (1985). ANIMENGINE: An Engineering Animation System. *IEEE Computer Graphics and Applications, October*, pp. 24-33.

Ostby, E. (1987). Survey of Computer Graphics for Character Animation. *Eurographics '87 Tutorial.*

Parke, F.I. (1982). Parameterized Models for Facial Animation. *IEEE Computer Graphics and Applications, November*, pp. 61-68.

Pintado, X. and Fiume, E. (1988). Grafields: Field-Directed Dynamic Splines for Interactive Motion Control. *Proc. Eurographics '88*, pp. 43-54.

Potmesil, M. and Chakravarty, I. (1983). Modeling Motion Blur in Computer-Generated Images. *Proc. SIGGRAPH '83. Computer Graphics 17(3)*, pp. 389-398.

Pueyo, X. and Tost, D. (1988). A Survey of Computer Animation. *Computer Graphics Forum 7*, pp. 281-300.

Reeves, W.T. (1981). Inbetweening for Computer Animation Utilizing Moving Point Constraints. *Proc. SIGGRAPH '81. Computer Graphics 15(3)*, pp. 263-269.

Reeves, W.T. (1983). Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics 2(2)*, pp. 91-108.

Reeves, W.T. (1985). Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. *Proc. SIGGRAPH '85. Computer Graphics 19(3)*, pp. 313-322.

Reynolds, C. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. *Proc. SIGGRAPH '87. Computer Graphics 21(4)*, pp. 25-34.

Reynolds, C.W. (1982). Computer Animation with Scripts and Actors. *Proc. SIGGRAPH '82. Computer Graphics 18(3)*, pp. 289-296.

Schlag, J.F. (1986). Eliminating the Dichotomy Between Scripting and Interaction. *Proc. Graphics Interface '86*, pp. 202-206.

Selbie, S. (1989). An Introduction to the Use of Dynamic Simulation for the Animation of Human Movement. *Proceedings of Computer Animation '89*, pp. 33-45.

Shelley, K.L. and Greenberg, D.P. (1982). Path Specification and Path Coherence. *Proc. SIGGRAPH '82. Computer Graphics 16(3)*, pp. 157-166.

Singh, B. Beatty, J.C. Booth, K.S. and Rhyman, R (1983). A Graphics Editor for Benesh Notation. *Proc. SIGGRAPH '83. Computer Graphics 17(3)*, pp. 51-62.

Smyrl, J.L. (1978). *An Introduction to University Mathematics*, Hodder and Stoughton.

Spackman, J. N. (1989). *The Use and Automatic Generation of Scene Decompositions for Accelerated Ray Tracing*, Ph.D thesis, University of Bath.

Spencer-Smith, T. and Wyvill, G. (1989). Four Dimensional Splines for Motion Control in Computer Animation. *Proceedings of Computer Animation '89*, pp. 153-167.

Steketee, S.N and Badler, N.I. (1985). Parametric Keyframe Interpolation Incorporating Kinematic Adjustment and Phrasing Control. *Proc.*

SIGGRAPH '85. Computer Graphics 19(3), pp. 255-262.

Stern, G. (1983). Bbop - A Program for 3-Dimensional Animation. *Proc. NICOGRAPH '83*, pp. 403-404.

Stroustrup, B. (1986). *The C++ Programming Language* Addison Wesley.

Thalmann, D. (1989). Motion Control: From Keyframe to Task-Level Animation. *Proceedings of Computer Animation '89*, pp. 3-17.

Thomas, F. and Johnstone, O. (1981). *Disney Animation The Illusion Of Life.* Abbeville Press.

Thomas, F. (1984). Can Classic Disney Animation be Duplicated on the Computer? *Computer Pictures 2(4)*, pp. 20-26.

Tost, D. and Pueyo, X. (1988). Human Body Animation: A Survey. *The Visual Computer 3*, pp. 254-264.

Wallace, B.A. (1981). Merging and Transformation of Raster Images for Cartoon Animation. *Proc. SIGGRAPH '81. Computer Graphics 15(3)*, pp. 253-262.

Waters, K. (1987). A Muscle Model for Animating Three-Dimensional Facial Expression. *Proc. SIGGRAPH '87. Computer Graphics 21(4)*, pp. 17-24.

Whitaker, H. and Halas, J. (1981). *Timing for Animation.* Focal Press Ltd

White, T. (1986). *The Animator's Workbook.* Watson-Guptill.

Wilhelms, J. and Barsky, B.A. (1985). Using Dynamic Analysis to Animate Articulated Bodies Such as Humans and Robots. *Proc. Graphics Interface '85*, pp. 97-104.

Wilhelms, J. (1986). VIRYA - A Motion Control Editor for Kinematic and Dynamic Animation. *Proc. Graphics Interface '86*, pp. 141-146.

Wilhelms, J. (1987). Dynamics for Everyone. University of California, Santa Cruz.

Wilhelms, J. (1987b). Toward Automatic Motion Control. *IEEE Computer Graphics and Applications, April*, pp. 11-22.

Witkin, A. and Kass, M. (1988). Spacetime Constraints. *Proc. SIGGRAPH '88. Computer Graphics 22(4)*, pp. 159-168.

Wyvill, B. McPheeters, C. and Garbutt, R. (1985). A Practical 3D Computer Animation System. *The BKSTS Journal*, pp. 328-332.

Zeltzer, D. (1982). Representation of Complex Animated Figures. *Proc. Graphics Interface '82*, pp. 205-211.

Zeltzer, D. (1982b). Motor Control Technique for Figure Animation. *IEEE Computer Graphics and Applications, November*, pp. 53-59.

Zeltzer, D. (1985). Towards an Integrated View of 3-D Computer Animation. *The Visual Computer 1*, pp. 249-259.

Appendix A

Three Dimensional Model Descriptions

This appendix contains an example of the object and lighting decription files required to model a three dimensional scene. Both the ray tracer and the mesh renderer use these description files. The object description given here is used to generate the head of the 'robot' character (see chapter 5).

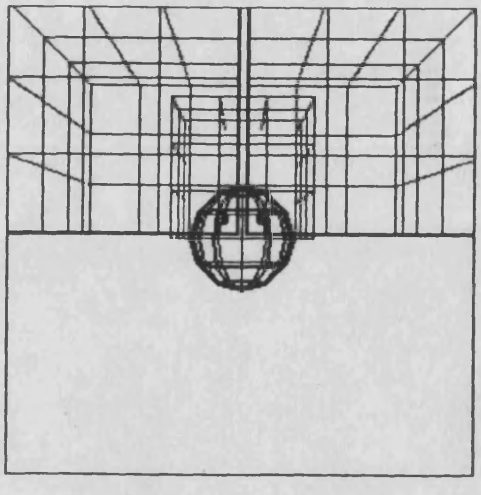
Note that the models are formed in a left handed coordinate system.

```
XMAX            256                 image resolution
YMAX            256
SCREEN          1.0 1.0 1.0         screen dimensions
VIEW            0.0 1.0 -6.0        location of view point
SCENE_CENTRE    0.0 1.0 0.0         location towards which the viewer points
SCR_ROT         0.0                 rotation of the screen
BCOLOR          0 0 0               background colour
FOG             0.0

LT_TYPE         1                   point light source
LT_INTENSITY    1.0
LT_ORIGIN       -8.0 10. -21
LT_COLOR        1.0 1.0 1.0

LT_TYPE         0                   infinite light source
LT_INTENSITY    1.0
LT_ORIGIN       8.0 10. -21
LT_COLOR        1.0 1.0 1.0
```

A Lighting Model

--

```
MATERIAL        red_metal
COLOR           1.0 0.0 0.0
SURFACE         1.0
REFLECTION      0.0
MIRROR          0.0
TRANSLUCENCY    0.0
ATTEN_RATE      0.0
REFRACTION      1.0

MATERIAL        black_eyes
COLOR           0 0 0
SURFACE         1.0
REFLECTION      0.0
MIRROR          0.0
TRANSLUCENCY    0.0
ATTEN_RATE      0.0
REFRACTION      1.0

MATERIAL        cornea
COLOR           1.0 1.0 1.0
SURFACE         1.0
REFLECTION      0.0
MIRROR          0.0
TRANSLUCENCY    0.0
ATTEN_RATE      0.0
REFRACTION      1.0

PRIMITIVE       head
NAME            SPHERE
CENTER          0.0 1.0 0.0
ROTATION        0x 0y 0z
SCALE           0.7

PRIMITIVE       head5
NAME            SPHERE
CENTER          0.0 1.0 0.0
ROTATION        0x 0y 0z
SCALE           0.675

PRIMITIVE       head2
NAME            CUBE
CENTER          0.0 1.8 0.0
ROTATION        0x 0y 0z
SCALE           1.6 1.6 1.6

PRIMITIVE       head3
NAME            CUBE
CENTER          .95 1.95 -1.4
ROTATION        0x 0y 0z
SCALE           1.8 1.8 2.0

PRIMITIVE       head4
NAME            CUBE
CENTER          -0.95 1.95 -1.4
ROTATION        0x 0y 0z
SCALE           1.8 1.8 2.0

PRIMITIVE       l_eye
NAME            SPHERE
CENTER          0.25 1.25 -0.5
ROTATION        0x 0y 0z
SCALE           0.075

PRIMITIVE       r_eye
NAME            SPHERE
CENTER          -0.25 1.25 -0.5
ROTATION        0x 0y 0z
SCALE           0.075

DISPLAY head2*(head-head5)-
        head3 - head4
MADE_OF red_metal

DISPLAY head2*head5
MADE_OF cornea

DISPLAY l_eye
MADE_OF black_eyes

DISPLAY r_eye
MADE_OF black_eyes
```



The Object Model used for the 'Robot' Character's Head

Appendix B

Colour Plates

The following colour plates are divided into two categories. The first four plates illustrate the Controller animation system in use. The final two plates are example frames taken from the robot animation, "Hit the Show, MAC!". They have been rendered using the ray tracer.

Plate 1: Controller's Title Page

Plate 2: A Cast Menu

Plate 3: Defining the Height Parameter along a Motion Path

Plate 4: Motion Paths from the Specification of the Robot Animation

Plate 5: The Robots soon after they have been Activated

Plate 6: The Procession of Robots Entering the Dome

Plate 1. Controller's Title Page

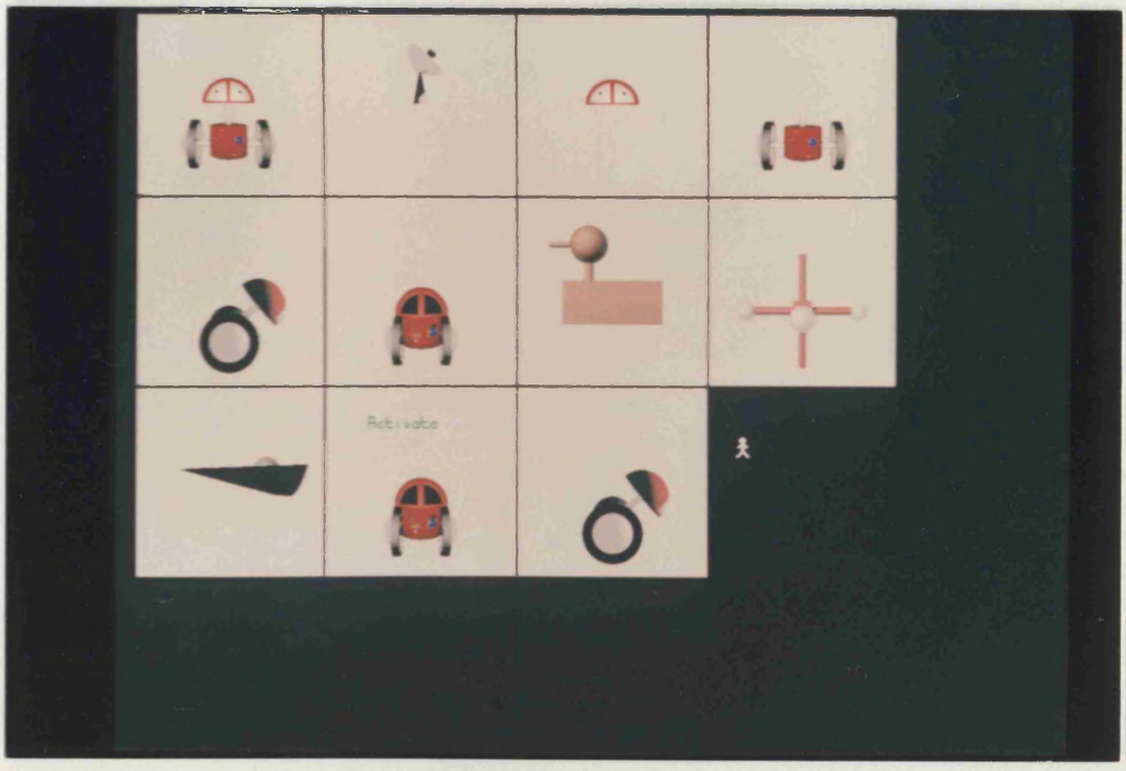


Plate 2. A Cast Menu

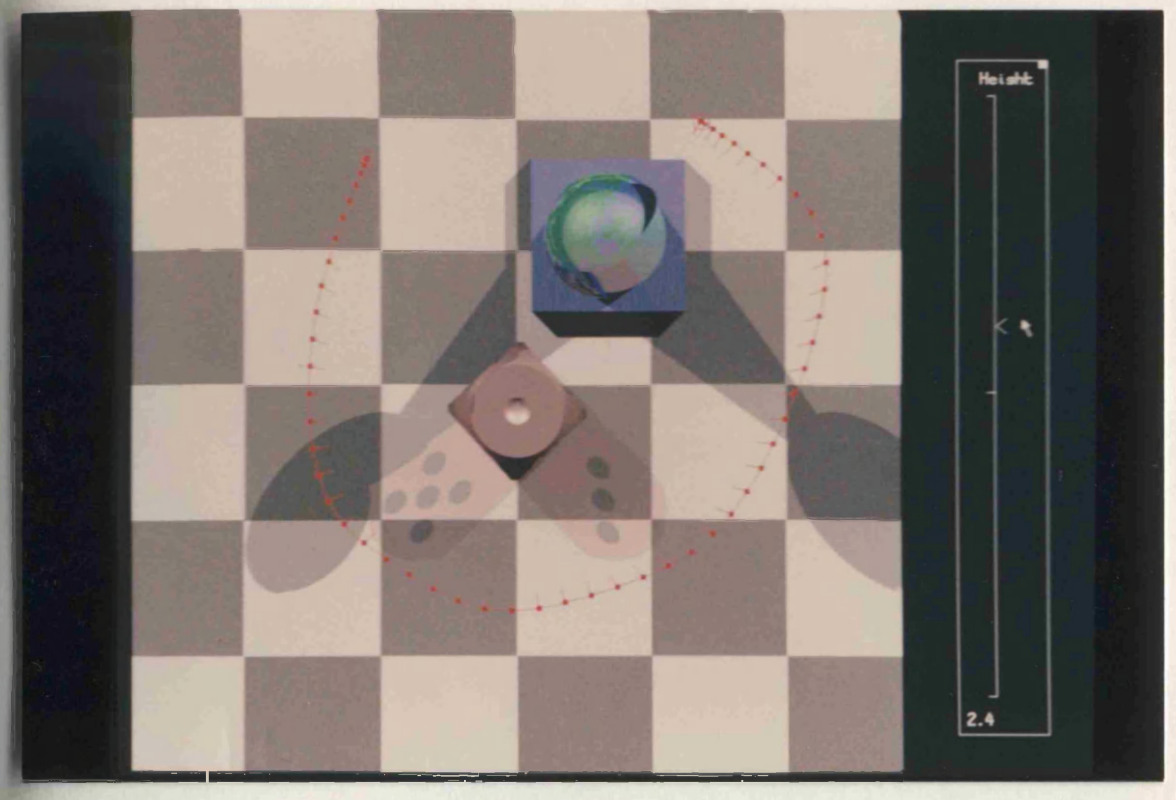


Plate 3. Defining the Height Parameter along a Motion Path

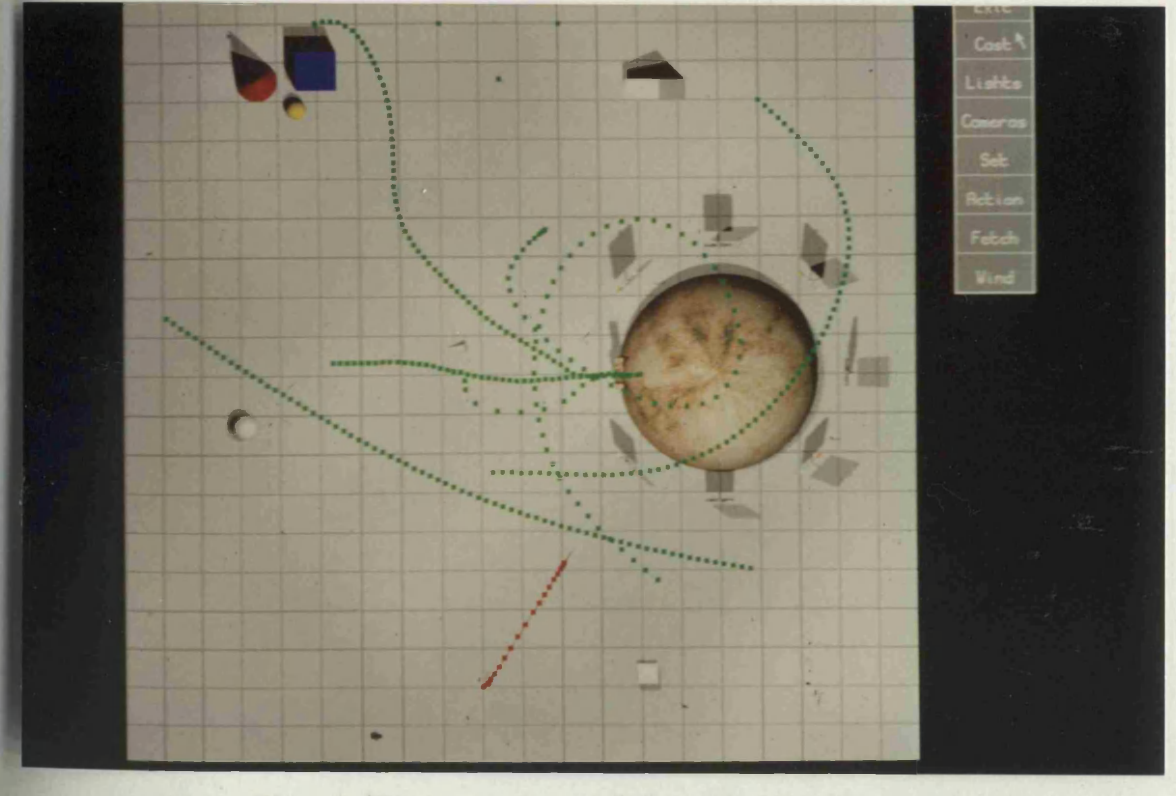


Plate 4. Motion Paths from the Specification of the Robot Animation

Plate 5. The Robots soon after they have been Activated

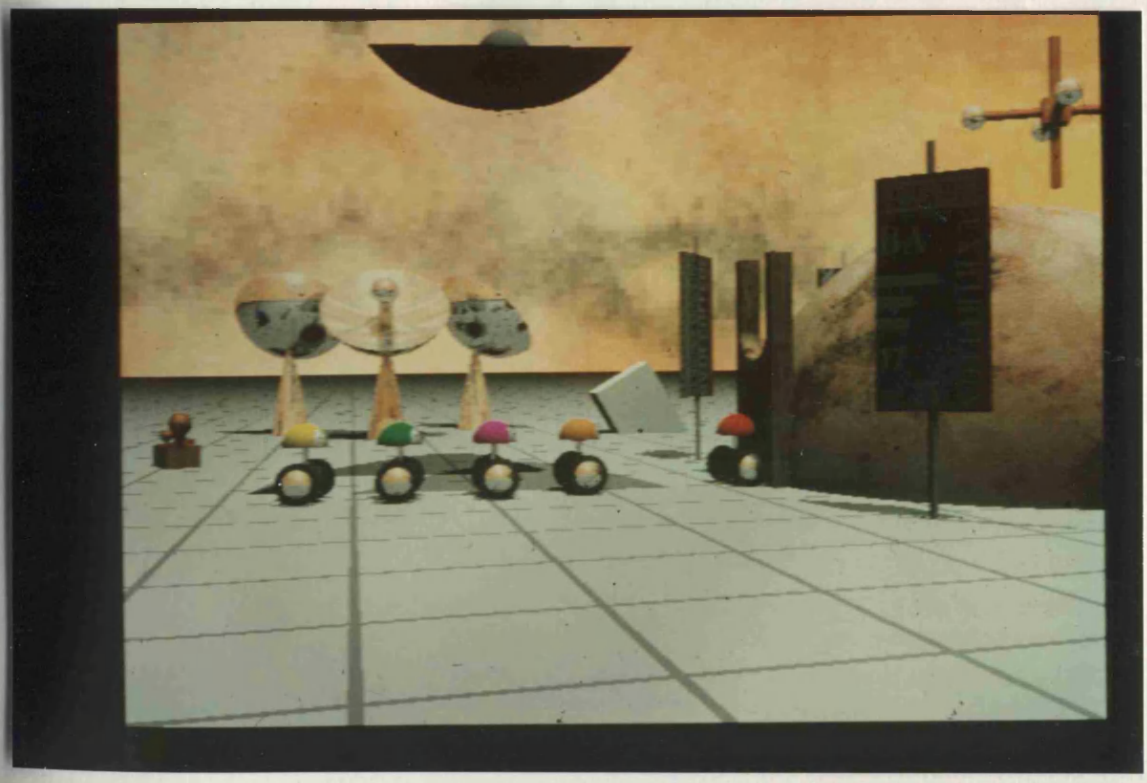


Plate 6. The Procession of Robots Entering the Dome

Appendix C

Extract from the Robot Storyboard

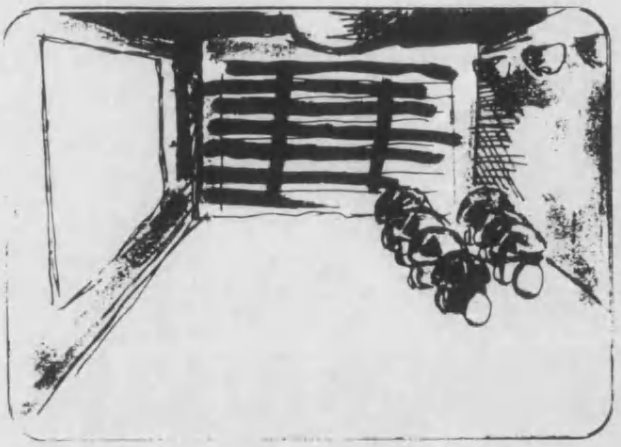
This appendix contains an example of the storyboard used during the production of the robot animation, ''Hit the Show, MAC!''.

Statistics

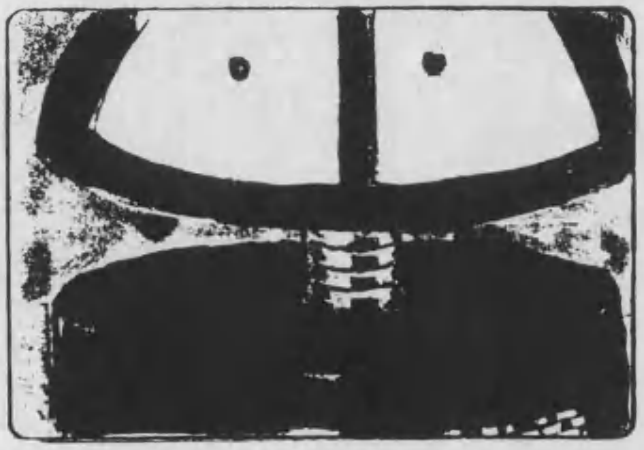
Animation length:	68 seconds
Frames generated:	420
Approximate rendering time per frame:	3 hours CPU time
Total CPU time for animation:	1,230 hours
Time spent on the production:	3 months

Credits

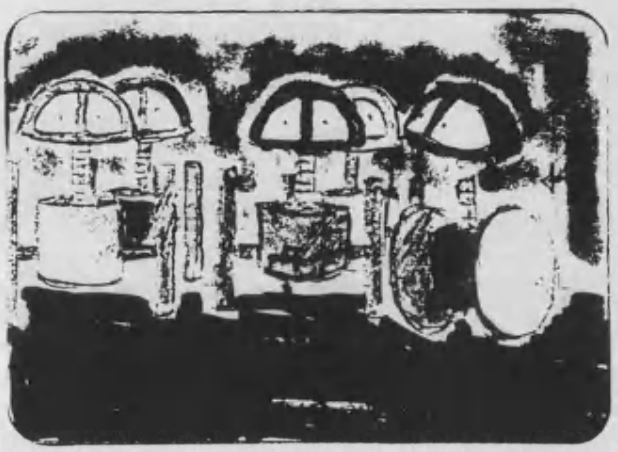
Designer:	Peter Ming Wong, Bath College of Higher Education
Producer:	Nigel John, University of Bath
Ray Traced Images:	John Spackman, University of Bath
Video Transfer:	CAL Videographics, London
Soundtrack:	HTV West, Bristol



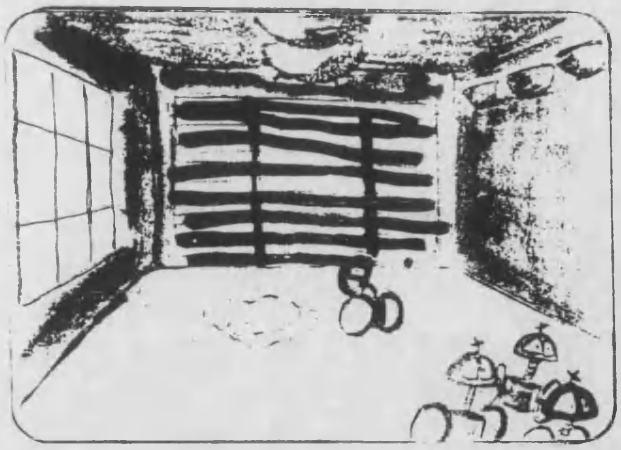
scene: *1*
action: *camera shot showing room and dormant robots;*
sound: *atmosphere.*



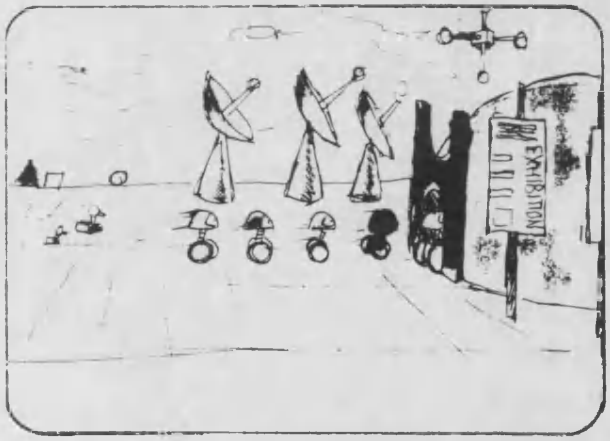
scene: *1*
action: *robots' heads raise;*
sound: *William Tell Overture, bleeps.*



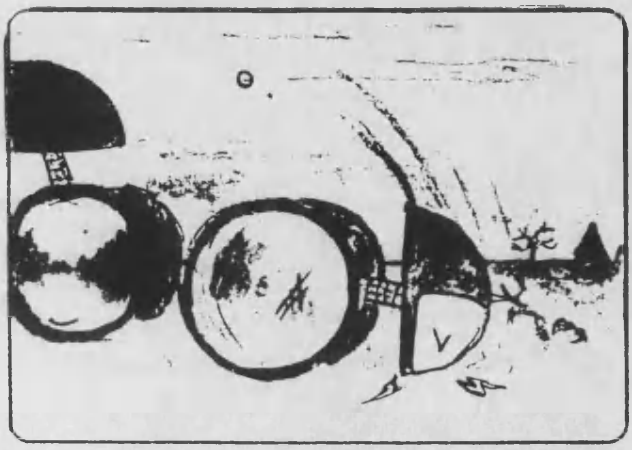
scene: *1*
action: *the chief inspects his subordinates;*
sound: *William Tell Overture.*



scene: *1*
action: *robots leave for the degree show;*
sound: *William Tell Overture, squeaky wheels.*



scene: *2*
action: *robot procession enters dome;*
sound: *William Tell Overture.*



scene: *2*
action: *faulty robot is knocked over;*
sound: *'crashing' noises.*

Appendix D

Publications

"The Controller Animation System"

This paper co-authored by P.J. Willis was presented at Eurographics(UK) in March 1987. It was published in *Computer Graphics Forum 8(2)*, June 1989.

"Some Methods to Choreograph and Implement Motion in Computer Animation"

This paper co-authored by P.J. Willis was presented at Computer Animation '89 in June 1989. It was published in *State-of-the-Art in Computer Animation*, Springer-Verlag, 1989.

The Controller Animation System

Nigel W. John and Philip J. Willis*

Abstract

One of the ways in which computer animation can be generated is to use interactive graphics systems. In this paper we describe Controller, an animation system of this type. We concentrate on the operation of Controller and detail how an animator can plan and choreograph the motion of objects and cameras. We have intended to provide a system that gives the animator as much control as possible subject to keeping the interface simple to operate.

1. Introduction

Computer animation systems fall into a number of different categories[1], with their own advantages and disadvantages. Typically there is a division between interactive systems and scripted systems producing animation in two or three dimensions. The methods used within one of these categories can also follow quite different approaches. For example, an interactive system could use either key framing techniques or path specification to develop motion.

In this paper we describe *Controller*, a system to produce three dimensional computer animation. We have concentrated here on a description of the user interface, but of equal importance have been the methods used to implement the facilities it offers. We decided to build an interactive system using path specification, with the intention of making it flexible and easy to use. However, why is another animation system needed at all?

2. Generating Motion

Whether we are concerned with traditional or computer generated animation we have to find ways of making objects move, yet different methods of producing motion have been developed within the two mediums.

In computer animation we are currently seeing a progression from the use of kinematics to the use of dynamics[1-3]. With kinematics some criterion is used to calculate the position of the moving object over time. However, more realistic results will be obtained if the motion is modelled using an appropriate physical law, such as one of the laws of dynamics. The cost is an increase in computation time, but there is also an increase in the amount of automation in the system (and arguably less fine control for the animator). In our animation system emphasis has been placed on giving the animator as much control as possible subject to keeping it simple to operate.

In contrast traditional animation is not always concerned with whether the motion required is possible in the real world. For example, when a cartoon character is chased over the edge of a cliff he does not fall immediately. He will run on in mid air for a few seconds before he realises that he is no longer on solid ground and only then will he fall. Here the animator is more interested in entertaining the audience, and therefore needs to make the motion look right in an artistic rather than in a scientific sense. Traditional animators have developed a number of principles to help achieve effects such as this. A summary of some of them follows and further details can be found in the literature[4-6].

Staging
Make sure that the action is well laid out and prevent the audience from getting confused.

Slow in and slow out
Space successive frames to make the moving object slow down or speed up to ensure smooth motion transitions.

Anticipation
Let the audience know what is about to happen by using a preparatory move e.g. swing a leg backwards before kicking a ball.

Timing
The timing depends on the number of drawings being used for an action. It can be used to emphasise the weight and size of a moving object.

Arcs
Motion will look less mechanical if the path of the moving object traces out a curve rather than a straight line.

Follow through and overlapping action
Make sure that an action does not end suddenly and determine if it will affect any subsequent action.

This paper was presented at the 7th Annual EUROGRAPHICS (UK) Conference, Manchester, March 29-31, 1989.

*Computing Group
School of Mathematical Sciences
University of Bath
Bath, Avon, UK

North-Holland
Computer Graphics Forum 8 (1989) 133-138

Secondary action
Enhance the main action with smaller secondary actions.

Squash and stretch
Deform a moving object in order to remove the appearance of rigidity.

Exaggeration
Exaggerating an action can help it appear more realistic, or at least caricature reality.

Note that not all of these principles conform to any physical laws. Greater use of these principles in computer animation has been advocated by Van Baerle[7], and Lasseter[4] and they have been a guideline in the design of Controller.

3. Designing an Animation System

We decided to model the interface of our animation system on a television control room. To determine the facilities that should be offered by our system we considered the activities which are monitored from such a control room.

Firstly the set for the scene to be filmed is built. Cameras, lighting and microphones must then be positioned and set up. Various test shots may be made to establish camera paths and actor position and movement. Cast and crew must learn their scripts so that they know what they should be doing during the scene.

When filming takes place the programme controller will be coordinating all movements, special effects, and giving instructions where necessary.

We chose this studio model with the intention of creating from it a user friendly interactive graphics system in which the animator becomes the programme controller. At this stage therefore the animator is primarily concerned with choreographing the action: we also have work in hand on the more specialised task of animating the individual characters but that is not reported in this paper.

Before proceeding to explain the facilities that Controller provides we briefly describe the equipment we have available.

4. The Graphics Environment

The Graphics Group at Bath University work with two Orion-1/05 super minicomputers running under the UNIX 4.2 BSD operating system. Other hardware includes two eight-bit colour displays with a resolution of 1280 by 1024 pixels, two digitising tablets with four-button pucks, a colour digitizer and equipment for automatic screen photography at 35mm still and 16mm cine.

Software in the group is written in the C programming language. It includes a library of graphics operations and a variety of tools used in conjunction with the above hardware. As well as providing the interface between the screen and the tablet, this library offers the usual drawing facilities, colour control, etc. We have a state of the art ray tracer and can also render meshes.

We thus have a highly interactive system with good quality raster scan colour displays, encouraging us to write interactive rather than batch-oriented programs.

5. Producing Animation Using Controller

5.1. Overview

Controller has been developed so that options are selected via menus, and parameters are set using valuator simulators (e.g. dials and scales). At the top level the animator can either accumulate animation data by dealing with set, cameras, lights or cast options, or convert the data into a scene model for each frame. Depending on his choice the appropriate sub-menu or valuator is displayed, and the system will wait for further interaction.

We will now look in more detail at the operation of each part of Controller, and consider some of the thinking behind it.

5.2. Choosing a Set

A number of simple scene models have been produced for experimental purposes. When the animator wishes to select a set a front elevation of each set is displayed on the graphics screen and the puck can be used to pick the one that is required. We prefer to use a front elevation because this gives a good feel for the appearance of the set. Once a set has been selected its plan view is displayed at a resolution of 1024 by 1024 pixels. It is on this plan view that most of the movement is planned by a combination of direct drawing with some mechanical assistance. The user also specifies the scene number and start frame number at this stage.

5.3. Cameras, Cast and Lights

In general an animator will select one of these items and then plan its movement. The process of selecting a cast member is similar to that of a set. For each new cast member a model is produced and an appropriate view rendered. These views are displayed on the graphics screen and the animator chooses the one he wants. Cameras and lights are numbered and so are selected from menu options, one entry for each camera and one for each light.

The animator can now begin to stage the action by defining a path around the set for the chosen camera, cast member or light.

5.4. Defining a Path

Firstly a path is plotted onto the plan view of the set. In general we have to define a three dimensional path using a two dimensional device, so we consider first only the xz plane. The extension into three dimensions by the setting of the object's height is described later. One of the principles of traditional animation explains that natural movement tends to follow arcs rather than straight lines. The use of B splines is ideal for this task as it produces a smooth curved path that can easily be altered locally. The animator thus draws a series of (rubber-banded) straight line segments onto the plan view of the set in order to approximate the desired path (Figure 1a). The end points of these segments are the points to be interpolated by the spline. These points are used to calculate the control vertices of the spline using a method described by Barsky and Greenberg[8].

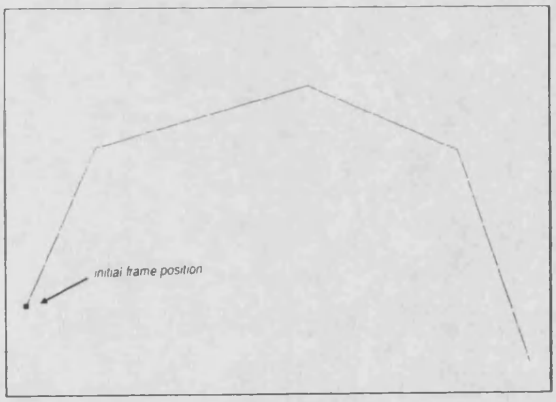


Figure 1a. Defining the shape of the path

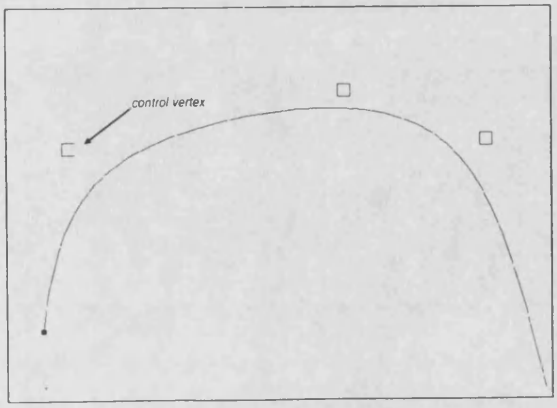


Figure 1b. The B spline path with its control vertices

The B spline curve is then calculated and drawn (Figure 1b). The animator can easily adjust the path by selecting a control vertex and moving it to a new position. Only the two spline segments immediately before and after the changed control vertex need be redrawn because each control vertex has only a local influence.

The option of using straight lines to define a path is still available to the animator as this is more appropriate in some cases. For example, a camera often tracks along a straight line.

So far we have only been generating positional information, without any reference to time. Therefore when the animator is happy with the path the next step is to decide where the object should be at each frame of the scene. This is an important stage in determining the realism of the motion achieved.

5.4.1. Motion Planning

Moving objects can accelerate, move at a constant speed, or decelerate. We have designed Controller so that these three modes of motion can be combined in any manner. It is left to the animator to dictate whether the result appears natural or unnatural.

The animator is presented with a menu of the three modes of motion. To specify a segment of motion he selects the mode required, and enters the duration of this segment in terms of the number of frames. The necessary frame positions are then calculated using the appropriate modelling function, and plotted onto the path. For example, in Figure 2a the animator has specified that the object should accelerate for ten frames. If the result is unsatisfactory it can be undone and another attempt made. Controller keeps

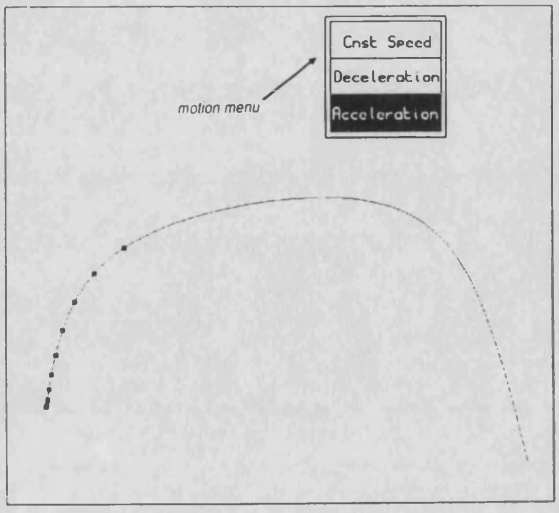


Figure 2a. Accelerating along the path

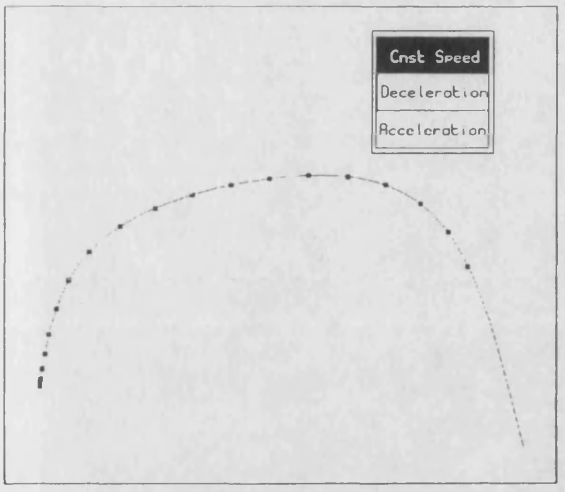


Figure 2b. Continuing along the path at a constant speed

track of the current speed attained by the object at the end of each motion segment and uses this to ensure that a smooth carry on is obtained at the start of the next motion segment (Figure 2b). The animator will be warned if the specification he requires is unrealistic. For example, he may ask for a stationary object to move at a constant speed without first accelerating it from rest. The warning can be ignored, however, as the animator may actually want such an effect. The timing of the motion by the animator is thus very flexible, and he can easily achieve effects such as slow in and slow out.

If the animator desires, Controller can automatically take into account the weight of an object. He has to inform Controller what the weight is (using an arbitrary unit of weight) and the modelling functions will then use this value to ensure that lighter objects accelerate and decelerate in less time than heavier objects.

Note that we are only defining the overall motion about the set. This is satisfactory for cameras and lights, but a member of the cast will also have its own internal movement, such as limb motion. Controller only takes a very simplified approach to achieve this at present. For a cast member enough poses are created to depict the motion style being exhibited and each pose will then be used in turn at successive frame positions. Figure 3 shows three poses of the pacman character used to depict his chomp.

5.5. Setting the Values of an Object's Parameters

A variety of parameters can be associated with each frame and can be made to change dynamically. As soon as the frame positions along the object's path have been determined parameter values can be specified at certain key frames and Controller will then interpolate across the in-between frames. The animator selects (on screen) a start and end frame position and gives the

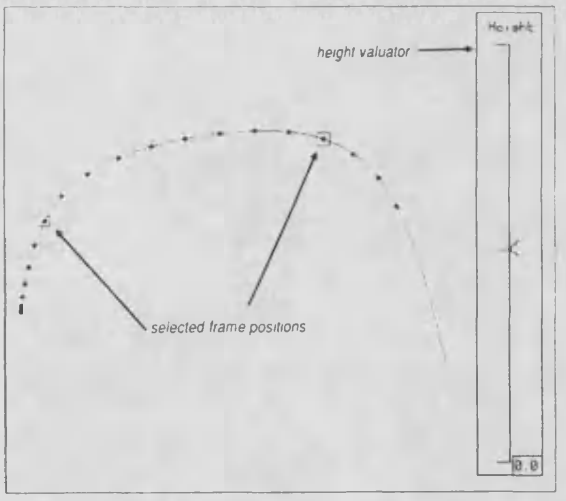


Figure 4. Setting the height

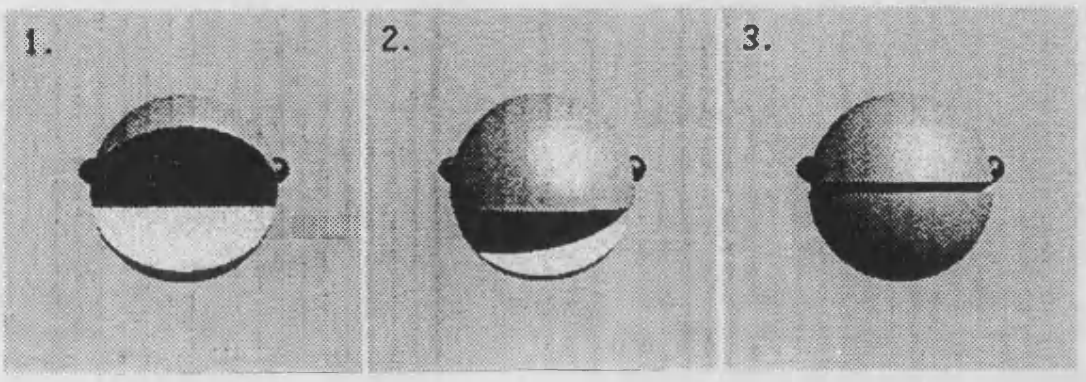


Figure 3. Motion poses

value of the parameter being set at these points. The in-between values are then calculated and assigned to the intermediate frames. It is just as important to consider traditional animation principles for this calculation as it was when the position along the path was being established.

For example, to complete the definition of a three dimensional path the height of the object must be set. A suitably scaled valuator is used to set the height at the end frame positions (Figure 4). The animator can then choose the criterion for calculating the height at the in-between frames from the following:

Linear
The total height difference is divided equally between all the frames involved.

Faired
The height difference is divided so that successive increments are increasing at the beginning, and decreasing at the end.

Gravity
Successive increments in the height are calculated as if the object is falling or rising under gravity.

Constant
All the in-betweens are assigned the same value.

Similar methods are used when the animator sets the value of other parameters. Hence a camera can be made to zoom by changing its lens angle, its heading can be set, and it can be tilted. Camera heading can be fixated on a particular point in the scene, it can be tangential to its motion, it can rotate or it can have a fixed heading regardless of motion. The heading of the cast and lights may also be set and the intensity of a light varied.

5.6. Using the Animation Data

The animator will need to preview the results of his planning and then make any necessary adjustments. To achieve this Controller offers a pencil test facility which presents the user with an animated line drawing on the screen. This test can be of either a cast member or of a camera. The animator uses the path on the screen to select a range of frames over which the pencil test is to be performed. If the object is a camera then the set view at each position along the path is calculated and displayed. At this stage none of the cast members are shown. If the object is a cast member it is also necessary to specify the camera it is to be viewed from. The sequence presented is that resulting from animating both this camera and the cast member for the selected frames. It shows the set and the selected cast member only.

At the end of a session all the information collected is written to a data file. The animator can load this file back into Controller in order to add further detail, or he can use it to generate a scene model for each frame of the film. During this frame generation stage the animator interacts with the system, coordinating and controlling all the data available. He has to decide which camera is to be used at each frame and to make sure that the cast are activated at the correct time. He also decides how the frames are to be rendered and at what resolution. The fast wire frame renderer can be used to preview the animation sequence before the more time consuming ray tracer is evoked.

Once rendering is complete it only remains to put the finished frames onto film or video.

6. Summary and Conclusions

We have described how an animator can use Controller to generate computer animation sequences. He should be able to apply the traditional animation principles and in many cases Controller will assist him in doing this. The animator should also find himself in complete control of what he wants to achieve.

We have been using Controller as a test bed for experimenting with computer animation techniques. It is intended to offer an alternative to the more complex systems that utilise the laws of dynamics. The basis of motion in our system is kinematics, but building on existing kinematic techniques in order to improve results. This is where the main thrust of the future development of Controller will be.

Acknowledgements

We would like to thank all the members of the graphics group for their help and suggestions, and the U.K. Science and Engineering Research Council for funding the project.

References

1. Jane Wilhelms, "Toward Automatic Motion Control," *IEEE Computer Graphics and Applications*, pp. 11-22 (April 1987).
2. Richard V. Lundin, "Motion Simulation," in *Nicograph '84 Proceedings*, Tokyo (1984).
3. Nadia Magenat-Thalmann and Daniel Thalmann, "Three-Dimensional Computer Animation: More an Evolution Than a Motion Problem," *IEEE Computer Graphics and Applications*, pp. 47-57 (October 1985).
4. John Lasseter, "Principles of Traditional Animation Applied to 3d Computer Animation," *ACM*

Computer Graphics (Proc. SIGGRAPH 87) **21**(4), pp. 35-44 (July 1987).

5. Frank Thomas and Ollie Johnstone, in *Disney Animation The Illusion Of Life*, Abbeville Press, New York (1981).
6. Tony White, *The Animator's Workbook*, Watson-Guptill, New York (1986).
7. Susan Van Baerle, *Character Animation: Combining Computer Graphics and Traditional Animation*, Eurographics 87 Character Animation Tutorial, August 1987.
8. Brian A. Barsky and Donald P. Greenberg, "Determining a Set of B-Spline Control Vertices to Generate an Interpolating Surface," *Computer Graphics and Image Processing* **14**, pp. 203-226 (1980).

Some Methods to Choreograph and Implement Motion in Computer Animation

NIGEL W. JOHN and PHILIP J. WILLIS

Abstract

Many methods of choreographing motion in computer animation have been developed. Many of the earlier key frame and scripted animation systems tended to require considerable effort from the user. With the development of systems using physical laws greater automation has been introduced, and more complex animation can be generated. The animator can argue however, that he is losing fine control over the motion produced. We wanted to develop a system that gives the animator as much control as possible over motion choreography, without the interface becoming too cumbersome to use. This paper describes some of the methods that we have used to achieve this aim.

Keywords: computer animation, faking mass, interactive, motion choreography, smooth motion.

1. INTRODUCTION

The art of animation is in making objects move in a convincing manner, which traditionally has depended on the skill of the animator. However, with the development of computer animation systems the resulting animation is also dependent on how the animation system allows motion to be choreographed.

Early computer animation systems often modelled closely the methods used to produce traditional animation, such as key framing (Catmull 1979). Although interactive they tend to require large amounts of input from the animator and are not always easy to use. An alternative is to use scripted systems that often appear in the form of an animation language. Although such systems have their advantages they tend not to be ideal when it comes to motion specification (Entis 1986).

Recently there has been research into developing more complex animation by using techniques such as dynamic analysis, automatic path planning, and stochastic algorithms (Magnenat-Thalmann 1985; Wilhelms 1987). Inherent with such systems is greater automation of the animation process. Arguably, greater automation can mean that the traditional skills of an animator are in danger of being overlooked.

There should be a place for the use of the traditional principles of animation in the computer medium (Van Baerle 1987; Lasseter 1987). We decided, therefore, to investigate ways in which computer aided motion choreography can be carried out. We have two goals to our approach:

(1) Giving the animator fine control over the motion produced;
(2) Keeping the interface simple.

2. THE ANIMATION TEST BED

Before we could experiment with the choreography of motion in computer animation we needed a system that would produce animated sequences. As we have already stated, an interactive system is generally considered to be preferable for motion definition tasks. We therefore implemented a system of this type, called *Controller* (John 1989), and a brief description of it is given here.

The overall interface of Controller is based on the operation of a television control room. Menus are used to drive the system and it makes use of graphical valuators such as dials and scales for the input of numerical data. When the animator wishes to film a scene he will be presented with a plan view of the set to be used. The animator is analogous with the programme controller and coordinates all the movements of the cameras, lights, and cast taking part. To choreograph the motion of one of these objects he will begin by planning the path that it will take in the xz plane of the set. To determine this positional information we have used a method of path specification similar to that described by Shelley and Greenberg (1982). We provide for a linear path, or a smooth continuous path (by using B splines). The latter can be adjusted if necessary. The next step is to determine the frame positions along this path. This is one way in which Controller differs from other animation systems and some of the methods it uses are described below. We also detail how the height of the object can be set over these frame positions.

Controller provides facilities for the setting of the object's orientation, changing the zoom of a camera, and varying the intensity of lights. The animator needs to know what effect changing one or more of these parameters has and should be allowed to make adjustments where necessary. Controller can, therefore, be instructed to display the view obtained by a particular camera at each frame, similar to a pencil test in traditional animation. A wire frame representation depicts the view obtained as this can be calculated in real time.

When the animator is happy with his specification all the data are converted into scene descriptions for each frame. Next, the animator decides when each camera will be used. The scene descriptions can then be fed to an appropriate rendering system, producing the final animated sequence.

3. MOTION IN COMPUTER ANIMATION

When an animation sequence is displayed it will be projected at a constant rate, video for example is projected at twenty five frames per second. Varying the time interval between successive frames in the sequence is not possible. Therefore an animation system has to calculate the position of a moving object at fixed time intervals. The distance covered during each of these time intervals will determine the overall effectiveness of the motion.

This section describes how the Controller animation system can be used to choreograph motion. We have attempted to provide a flexible interface offering the animator fine control over motion specification. However, we do not want to make this interface impossible to use; it should appear to the animator as a natural way of defining motion. The following sections detail some of the methods we have used to carry out the motion specification. In doing this we found that the ease of use of the interface decreases as the complexity of the problem increases. We have endeavoured, therefore, to keep the implementation as simple as possible.

3.1 Choreographing Motion

In Controller, the choreography of a moving object consists of two stages:

(1) Drawing a path to define the overall position of the object;
(2) Deciding on the object's position along the path at each frame.

We have already mentioned how an animator can do the first stage by using a path specification technique. An example of a path that has been drawn using B splines can be seen in Fig. 1. At this stage no regard has been given to time, this being the next stage of the problem.

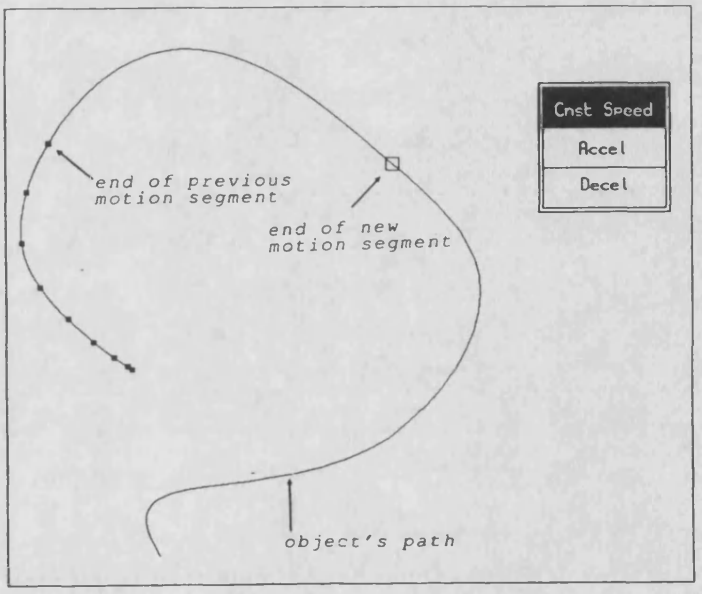


Fig. 1. Defining a Motion Segment

At present we do not provide facilities for modelling internal movement, such as the limb movement of a human character. Post process animation algorithms such as those described by Lundin (1984) would be ideal here. We are concerned with the overall motion choreography about the set. The animator divides the path into a series of *motion segments*, when the object will be accelerating, decelerating, or moving at a constant speed. These three modes of motion can be combined in any manner, the animator decides whether the resulting motion appears realistic or unrealistic. Sometimes he will be warned of an unrealistic combination. For example, he may ask for a stationary object to move at a constant speed without first accelerating it from rest. The warning can be ignored however, since the animator may really want such an effect.

To specify a motion segment the animator first shows how far he wishes the object to travel. This is the distance from the most recently calculated frame position to some point he now marks on the path (see Fig. 1). He then selects the motion style required from the menu displayed. If either acceleration or deceleration is selected, the animator is required to enter the number of frames to be taken. Using this specification the animation system will then calculate the frame positions along the path

segment. The procedure is slightly different if the object is to move at a constant speed; in this case the animator only needs to show the distance of the motion segment, and as many frames as possible will be fitted in. If the resulting motion segment is unsatisfactory it can be undone and another attempt made. He can also make the object remain at the most recently calculated frame position for any number of frames.

The animator should find that moving an object around its path is as natural as driving a car along a road. For the latter decisions about whether to slow down, speed up or stop depend on the road conditions and the driver's destination. In the animation case the decisions are similarly governed by the story board that describes the whole scene.

As well as defining motion segments, the animator can set various parameters over a range of existing frame positions. One of these parameters is the height of the object. A valuator such as a sliding scale sets the height at two frame positions. The animation system will then calculate the height at the frames between these, according to the animator's needs. The animation of this height change is also important.

3.2 Methods of Implementing Smooth Motion

An object in motion will be accelerating, decelerating, or moving at a constant speed. Our aim is to simulate such motion around an object's path using simple kinematic techniques. To produce realistic results we rely on the animator's skill and the flexibility of the system. We have avoided the use of splines and complicated physical laws. The latter tend to produce more realistic results but are also computationally more expensive. Further, both techniques increase the difficulty the animator has in specifying the exact motion he requires.

We use the duration of the fixed interval between successive frames as our unit of time when calculating motion. As we have seen, the animator places two constraints on each motion segment he defines in the ground plane:

(1) The length of the path segment;
(2) The number of frames to be taken.

We have to consider the interchange between successive motion segments. Usually we will require this interchange to occur smoothly.

Method 1: Trigonometric Functions

In key frame interpolation acceleration and deceleration effects are often modelled using

$$1 - \cos(t), \quad 0 \le t \le \pi/2;$$

$$\sin(t), \quad 0 \le t \le \pi/2$$

respectively (Magnenat-Thalmann 1985). These formulae can be used to calculate the frame positions along a path. We require

$$l(t) = \textit{length of path segment} \times (1 - \cos(\pi/2 \times t)), \textit{ or}$$

$$l(t) = \textit{length of path segment} \times \sin(\pi/2 \times t),$$

where l(t) is the fraction of the path segment length covered up to time t. The value of t is scaled so that it falls into the range [0,1] by using

$$t = \frac{\textit{current frame of this motion segment}}{\textit{duration in frames of this motion segment}}.$$

As well as accelerating or decelerating the object, the animator may require it to continue at a constant speed. This speed will be that

attained by the object at the end of the motion segment last calculated. A reasonable approximation of this is to take the average speed of the object between the two most recently calculated frame positions. This is, in effect, the distance between these two frame positions.

The overall motion definition will consist of some combination of motion segments. As an example suppose that the animator defines a sequence of motion segments during which an object:

1) accelerates from rest,
2) maintains a constant speed,
3) accelerates again,
4) decelerates.

The graph of distance against time for this motion definition is given in Fig. 2a. We require a smooth interchange between each motion segment. Note that if two or more successive motion segments are of the same style, acceleration for example, then the best results are obtained by combining them into one motion segment. We thus have less interchange points to consider.

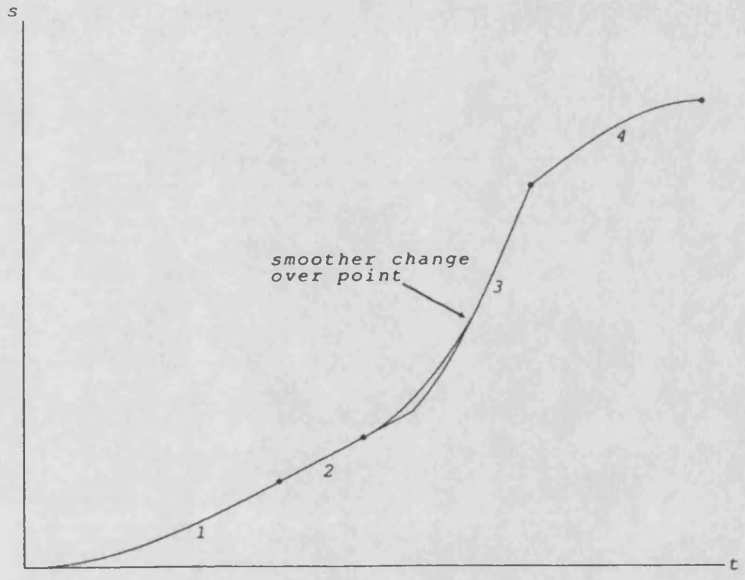


Fig. 2a. Using the Trigonometric Functions

The first motion segment of our example presents no problem: we fit the distance and time constraints to a graph of (1-cosine). When the object is not accelerating from rest, as in the third motion segment, this is not as easy to do as we have to take the current speed of the object into account. The acceleration function as it stands does not do this. We decided to adopt the simplest solution to this problem.

The distance yielded by the acceleration function is only used if it is greater than the distance that would be covered by the object continuing to move at its current speed. Until this happens the object's speed is not altered. The point at which the acceleration function takes over can be quite noticeable, as it is in Fig. 2a. However, if we increment the

object's speed after every frame by some appropriate amount so that the object does indeed appear to accelerate, then the change over point will be much smoother (Fig. 2a.). Note that up until this change over point the object's speed is increasing in an arithmetic progression. A similar technique can also be used when modelling deceleration with the sine function. In this case we have to ensure that the speed of the object is always less than its speed coming into the current motion segment. We have not needed to do this in the above example as the initial speed obtained from the sine function is a lot less than the speed of the object at the end of the third motion segment. In fact the difference is too great for a smooth interchange.

Sometimes the constraints that the animator has defined make it impossible to achieve the desired smooth interchange between successive motion segments. We can check for such cases and warn the animator who may then decided to change his motion definition.

Method 2: Laws of Motion for Constant Acceleration

Another way of modelling the motion effects we require is to use the laws of motion for constant acceleration. Particularly appropriate to our needs is the following motion law

$$s = ut + \frac{1}{2}at^2,$$

where s is the displacement, u is the initial speed, a is the acceleration, and t is the time. By substituting the animator's defining conditions into this equation we obtain the value of the acceleration required over the motion segment. The individual frame positions of the moving object can then be calculated. Figure 2b. depicts the distance against time graph obtained by using this technique on the same motion definition as in the previous method. We can now automatically get a good interchange between motion segments as this equation allows for the initial speed of the object.

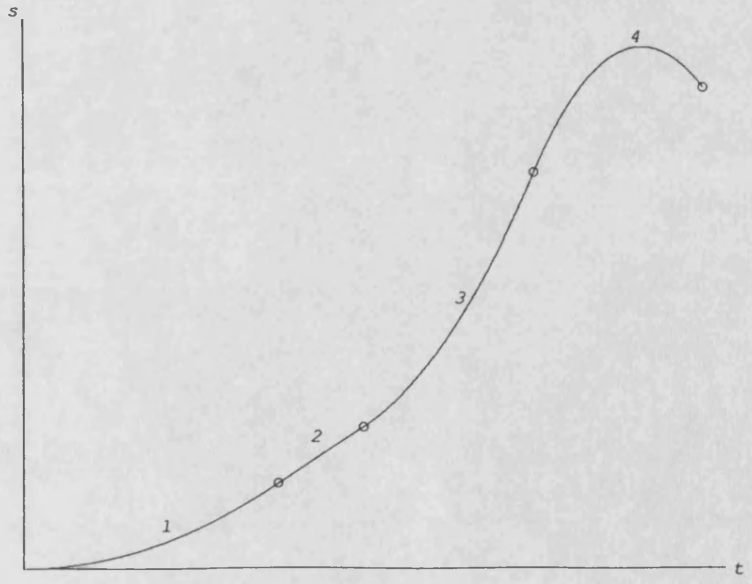


Fig. 2b. Using One of the Laws of Constant Acceleration

There is a further constraint, however, in that since the object only travels around its path in one direction, the value of the object's velocity must not change its sign during the motion definition. We found that when decelerating an object, in too many cases the animator's constraints could only be satisfied if the velocity did change sign. This is the case in the fourth motion segment of our example, the distance-time graph goes through a maximum. The motion segment in these cases had to be rejected.

3.2.1 Comparisons

For the purpose of comparing the above two methods we have superimposed the two distance against time graphs obtained (Fig. 2c.).

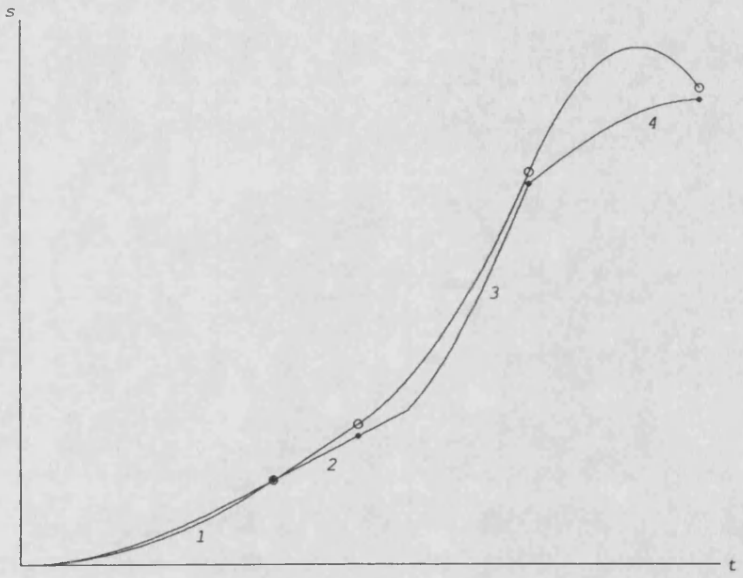


Fig. 2c. Comparison of Methods 1 and 2

We can see that both methods give acceptable results when accelerating an object from rest. The final speed attained by the object does differ, but this does not matter. We are not concerned with the actual value of the object's speed, only that the final motion is visually acceptable to the animator. Maintaining an object at a constant speed is straightforward in both cases. However, the law of constant acceleration is far better at accelerating an object when it is already in motion. The initial speed of the object is automatically taken into account so we do not need to do apply extra techniques in order to get a smooth interchange. As we have seen, however, there can be a problem when using method 2 to decelerate an object. In some circumstances the object can only satisfy the animator's constraints if it overshoots its destination and then comes back. We will not get this problem using the sine function, however. Fitting the motion to a sine curve ensures that the distance and time constraints are satisfied without the motion changing direction. The cost is that the initial speed of the object is ignored and so we lose out on the smoothness of the interchange.

Our best approach for achieving smooth motion has been to use a combination of the above methods. Priority is given to the law of constant acceleration, but where this fails the trigonometric functions are utilised. More complicated strategies would produce more realistic motion. However by using the techniques described above we get a quite acceptable method of motion planning that is straightforward both to implement and to operate.

3.3 Techniques For Faking Mass

By carefully timing the motion of an object an animator can emphasise its size or weight (White 1986). He has to make objects move more slowly as they get heavier, and perhaps give them more difficulty in controlling their weight.

Using the previous methods of motion planning the animator can easily make two objects move at different rates. If both objects are to move from rest, the first motion segment of each will be an acceleration phase. At any given time during this phase the total distance covered by the heavier object must be less than that covered by the lighter object. So if he wants both acceleration phases to last for the same amount of time he must ensure that the heavier object will traverse less of its path. This distance is one constraint that the animator has to define when specifying a motion segment, so there is no difficulty in doing this. If the objects now proceed at a constant speed the lighter object will be travelling at a faster rate. The animator's judgement in indicating the distance of the acceleration phase will determine how convincing the final result will be.

One aim of our approach to computer animation has been to give the animator fine control such as he has in the above example. We do not want to make the specification too difficult for the animator and it will not be if we are only concerned with a few objects. However, if the scene is to contain many objects each with a different weight, such motion control could become a headache to the animator. He has to keep track of how heavy all the previous objects were and fit in each new object accordingly. It might be easier if the animator just estimates the weight of each object and lets the animation system take care of the rest. We therefore provided a facility to do this.

When the animator selects a cast member he has the option of defining its mass. The unit of mass is immaterial as we only need to depict the relative mass of the objects in the scene. We now have to satisfy three user-defined constraints (mass, distance and time), so the complexity of the problem will increase. However, the purpose of this exercise is to save the animator from having to remember all the distances he has been using to emphasise the mass of each object. So we let the motion modeller work out how much of the path will be traversed and merely get the animator to specify time and mass.

Method 3: Utilising Existing Methods

When we use

$$(1-\cos(time)) \quad on\ [0, \pi/2]$$

to model distance travelled under acceleration the result is scaled by the total length of the motion segment. This segment length must now be determined by the system using a function that depends on the mass of the object and the total time of the motion segment. The same is true if we are using the law of constant acceleration, the segment length must now be calculated by the system and not defined by the animator. The segment length should increase with time but decrease as the mass of the object increases, so

$$\textit{segment length} \propto \frac{\textit{time}}{\textit{mass}},$$

The simplest relationship satisfying this condition is

$$\textit{segment length} = k \times \frac{\textit{time}}{\textit{mass}}, \quad k \textit{ constant}.$$

We let the animator determine the value of k by getting him to define the distance he would expect an accelerating object of unit mass to cover in some specified time interval. This task is performed by the animator before the path planning stage of our system. He uses a graphical valuator to input the required distance. The animator is thus still in general control of the final motion effects achieved.

Method 4: Using a Family of Acceleration Curves

Several other ways of modelling the motion of objects with different mass exist. For example, the function

$$y = \frac{x^2}{ax^2 + b}, \quad a \text{ and } b \textit{ are constants},$$

has been experimented with. By varying the values of a and b a family of curves can be produced (Fig. 3a. and Fig. 3b). We tried

a) varying the value of a whilst keeping b fixed,
b) varying the value of b whilst keeping a fixed.

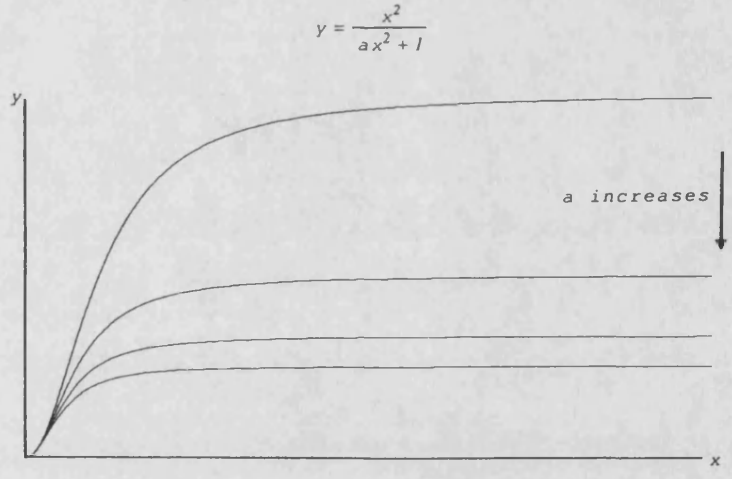


Fig. 3a. Changing the Value of *a*

We then considered what the motion effect would be if these graphs represented plots of velocity against time i.e.

$$v = \frac{t^2}{at^2 + b}, \quad v = \textit{velocity}, \; t = \textit{time}.$$

Each plot has the same general shape. Up to the point of inflection the moving object can be thought of as overcoming its inertia. It then proceeds to accelerate, tending toward some maximum velocity. Increasing the value of a decreases the maximum velocity obtainable by the object.

$$y = \frac{x^2}{x^2 + b}$$

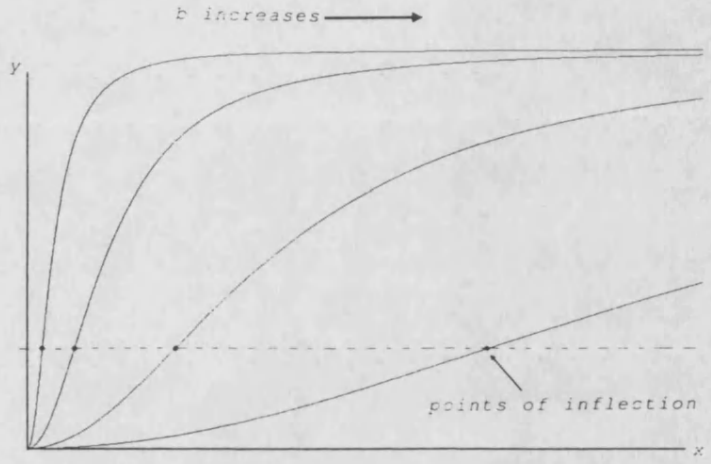


Fig. 3b. Changing the Value of *b*

This suggests that *a* is related to the force that propels the object. If *a* is fixed i.e. each object is propelled by the same amount of force, then increasing the value of *b* increases the time taken for the object to reach any given velocity - up to the maximum velocity obtainable. So *b* appears to be related to the weight or mass of the object.

We can get a distance against time function by integrating with respect to time

$$s = \frac{t}{a} - \frac{\sqrt{b}}{a\sqrt{a}} \tan^{-1}\left(\sqrt{\frac{a}{b}} \times t\right), \quad s = \textit{displacement}, \; t = \textit{time}.$$

We need to determine suitable values for *a* and *b*.

In our current implementation we emphasise the mass of the object so we keep *a* fixed at *a=1*. We do not set *b* to be equal to the value of the object's mass. The mass defined by the animator is typically in the range (0,100], but to obtain the best results we found that the value of *b* should be much larger. If *b* is too small then the object rapidly reaches its maximum velocity, too rapidly to be usable in our system (Fig. 4b.). We have found that multiplying the mass by a factor of 100,000 provides a suitable value for *b*. We again let the animator determine the value of the distance scale factor. He specifies the total distance that an object of unit weight will cover over a specified interval of time.

We also have to allow for the deceleration of an object. To do this we keep the time argument used by the distance function separate from the total time elapsed. The latter is always incremented as new frames are defined by the animator. We only increment our distance function time, however, when the object is accelerating. When the object is decelerating it is decremented, and when the object is moving at a constant speed it is left unaltered. We thus travel back down the distance time curve when decelerating and a correspondingly smaller inter frame distance is

calculated. Note that this method will give a smooth interchange between successive motion segments. There is a drawback, however. The object's rate of deceleration will be the same as its rate of acceleration, and so the animator cannot make an object decelerate over a period longer in time than it accelerated in. We can overcome this by varying the time intervals used with the distance function, but will lose out in the smoothness of the interchange between motion segments.

3.3.1 Comparisons

We want to compare the motion of objects that have been given different masses. As an example, consider the case when objects are accelerating from rest. Some distance-time graphs obtained by doubling the mass of an object whilst keeping the other constraints fixed are given below.

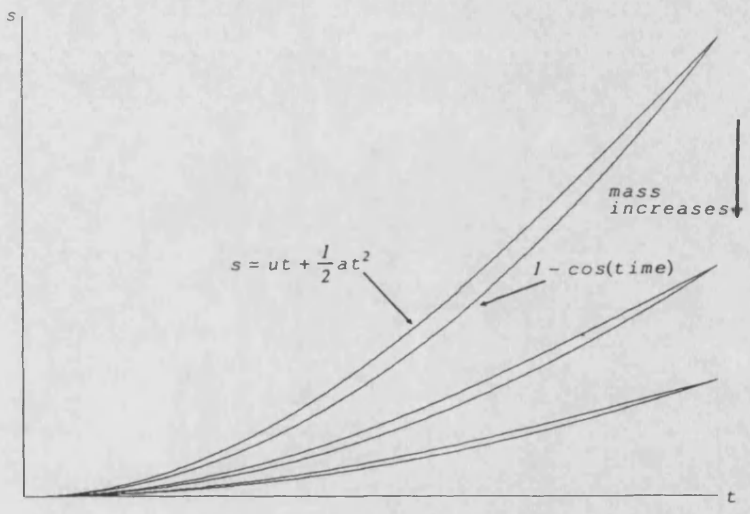


Fig. 4a. Using Existing Techniques

In Fig. 4a. we have used the techniques of method 3, that is the law of constant acceleration and the trigonometric function (1-cos). The animator has indicated the distance an object of unit mass is required to cover, the system will then automatically calculate other distances. We can see that in both cases the total distance covered in a fixed interval of time is halved as the mass of an object doubles. This is what we would expect if the objects are accelerating from rest. Repeating the procedure for method 4 yields the distance-time graphs of Fig. 4b. Again the distance covered appears to be halved as the mass of an object doubles. In fact this is not quite true, but when large values are used for b (as in our implementation) the differences are not significant.

Figure 4c. enables us to compare the shape of the motion graphs obtained by using method 4, and the law of constant acceleration from method 3. To aid in this comparison we have made sure that the final distance covered is the same in both cases. We can see that when we use method 4, an object accelerates more slowly to begin with, and consequently gives

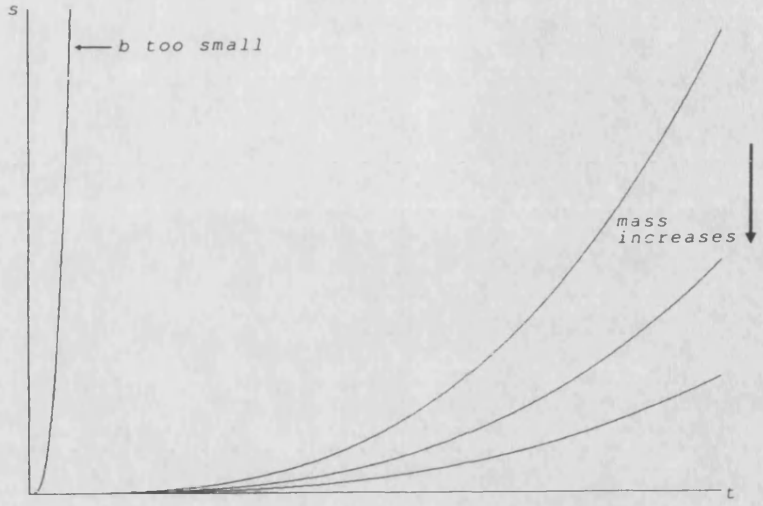


Fig. 4b. Using a Family of Acceleration Curves

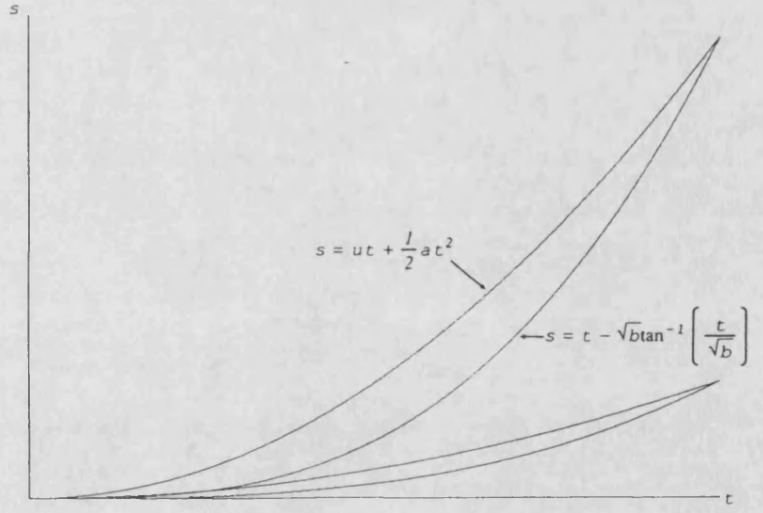


Fig. 4c. Comparison of Methods 3 and 4

us a better feel for the inertia of the object. It thus has to reach a greater final speed in covering the same distance in the same time interval as that of method 3.

We have preferred to use the family of acceleration curves method of incorporating mass. A motion graph resulting from this method is an intuitively better approximation to the real thing. We also have a way of manipulating the force applied to an object built into this distance function. This facility has not yet been exploited.

3.4 Height and Orientation

To complete the specification of a three dimensional path the height of the object must be determined. As described earlier, the animator will set the height at two frame positions and leave the system to calculate the height at the frames between. The animation of any height change is just as important as the determination of the object's position in the ground plane. We again leave it the animator to combine these two motions in a sensible way, but provide some tools to assist him.

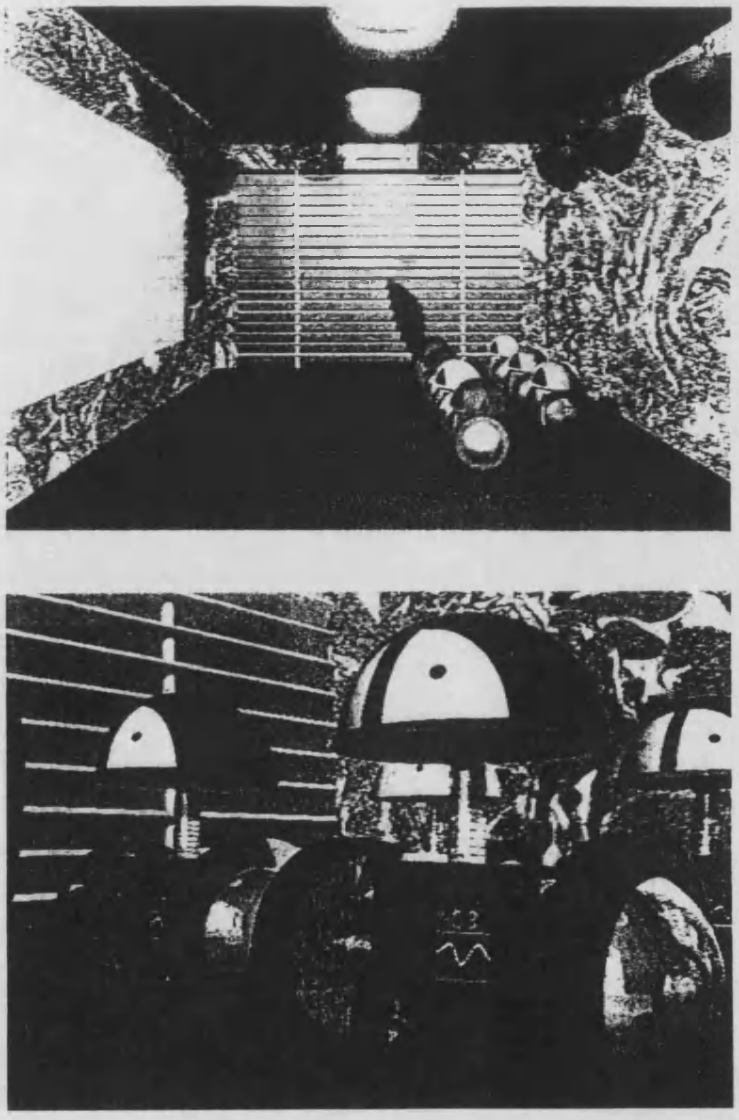
We know the height at the two frame positions the animator has set and thus the difference in height between them. The simplest way to calculate the height at each frame in between is to linearly divide this total height difference. This is analogous to an object at rest instantaneously reaching a constant speed without accelerating first. To give a smoother look to the height change the animator may be better off using a fairing technique. Such a technique will give a gradual increase in successive height differences at the beginning of the movement, and a gradual decrease at the end. We can use

$$1 - \cos(x) \quad on\ [0, \pi]$$

to model such a change. Alternatively, if a period of constant change is required at the centre of the movement we can use the technique of sinusoidal fairing described by Kingslake (1986).

What if the animator wishes the object to appear to be falling or rising under gravity? Then the object should be accelerating if falling, and decelerating if rising. Clearly the methods we have already used for modelling acceleration and deceleration should also be applied to the animation of height changes. Not only height changes can be treated in this way. We also have to animate changes in the object's orientation and the zoom setting of a camera.

As an example of changing the orientation of an object, let us look at how it can be made to stagger or vibrate. We require the object to swing back and forth for several oscillations. The amplitude reached on each swing will gradually decay until the object comes to rest at its equilibrium position. An animation system using the laws of dynamics would model such motion on a damped oscillation, but we take a simpler approach. The animator will define the total number of frames to be used, the number of oscillations required, and the maximum angle from its equilibrium position that the object can reach. As the object ends at rest and the amplitude of the extreme positions is decaying, fewer frames are assigned to each successive swing. We can use a deceleration function to determine how the total number of frames should be split up between the number of oscillations required. To give more *snap* to each swing an animator will usually require that more in betweens are used coming out of an extreme than are used going into it. An acceleration function should be used, therefore, to determine the position of the object between its extremes. All that remains to be done is to decay the angle of the extreme positions until the equilibrium position is reached. This can be done linearly or by using one of our other methods, according to the animator's wishes.



Two Frames from an Animation Sequence Currently Being Produced using Controller

4. SUMMARY AND CONCLUSIONS

We have presented some methods by which motion can be choreographed in a computer animation system. By keeping the modelling functions as simple as possible we have been able to keep the interface straight forward to use. We feel that this helps to present the animator with fine control over motion definition. He can thus apply traditional techniques to produce convincing animation.

ACKNOWLEDGEMENTS

We would like to thank all the members of the graphics research team at the University of Bath for their help and suggestions, the U.K. Science and Engineering Research Council for funding the project, and Peter Wong of the Bath College of Higher Education for his artistic input.

REFERENCES

Van Baerle S (1987) Character animation: combining computer graphics and traditional animation. Eurographics 87 Character Animation Tutorial: 134-145.

Catmull EE (1979) New frontiers in computer animation. American Cinematographer 60(10):1000-1003.

Entis G (1986) Computer animation - 3D motion specification and control. SIGGRAPH '86 Tutorial Notes.

John NW, Willis PJ (1989) The controller animation system. To be presented at Eurographics UK Conference 1989, to appear in Computer Graphics Forum.

Kingslake R (1986) An introductory course in computer graphics. Chartwell-Bratt, pp 100-102

Lasseter J (1987) Principles of traditional animation applied to 3d computer animation. Proceedings of Siggraph, Computer Graphics 21(4): 35-44

Lundin RV (1984) Motion simulation. Nicograph '84 Proceedings, Tokyo

Magenat-Thalmann N, Thalmann D (1985) Three-dimensional computer animation: more an evolution than a motion problem. IEEE Computer Graphics and Applications: 47-57

Magenat-Thalmann N, Thalmann D (1985) Computer animation theory and practice. Springer-Verlag, Tokyo Berlin Heidelberg New York, p49

Shelley KL, Greenberg DP (1982) Path specification and path coherence. Proceedings of Siggraph, Computer Graphics 16(3): 157-166

White T (1986) The animator's workbook. Watson-Guptill, New York, p74

Wilhelms J (1987) Toward automatic motion control. IEEE Computer Graphics and Applications: 11-22

Nigel John

Nigel John is a postgraduate student in the Computing Group at the University of Bath, where he is currently researching for the degree of *Doctor of Philosophy*. His research interests are mainly in the area of computer animation, particularly the choreography of motion in this medium.
Address: Computing Group, School of Mathematical Sciences, University of Bath, Bath, Avon, UK.

Philip Willis

Philip Willis is Head of the Computing Group at the University of Bath where he leads one of the larger university computer graphics research teams in the UK. This is active in applications of colour raster graphics to animation, workstation design, printing and picture archives, He is also a member of the Eurographics Conference Board, co-convener of Working Group 2 on Picture Databases and is Treasurer to the Eurographics UK Chapter.
Address: Computing Group, School of Mathematical Sciences, University of Bath, Bath, Avon, UK.